

Copy No. _____

52-054614 REVISION B

ATAC-16M

Principles of Operation

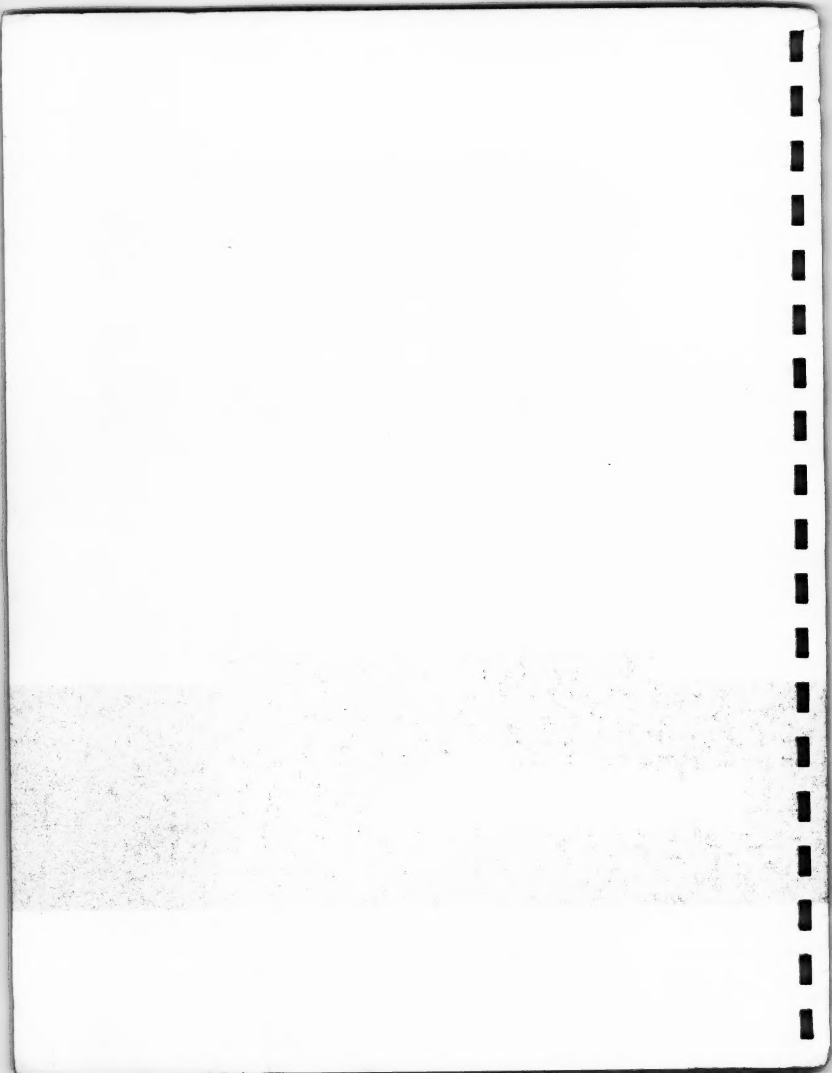
ATAC

APPLIED TECHNOLOGY'S ADVANCED COMPUTER



Applied Technology
A Division of Ittek Corporation

JUNE 1979



Copy No. _____

52-054614 REVISION B

al Puris

Itek

ATAC-16M

Principles of Operation

ATAC

APPLIED TECHNOLOGY'S ADVANCED COMPUTER

NOTICE

This publication is intended to inform the reader regarding the operational characteristics, physical construction, and capabilities of ATAC equipment and software. The contents of this publication should not, however, be considered an equipment or software specification, and Applied Technology in no way warrants the accuracy or completeness of this document for procurement purposes.

JUNE 1979

**COPYRIGHT 1979, ITEK CORPORATION
COMMON LAW COPYRIGHT RESERVED**

Applied Technology

A Division of Itek Corporation

645 Almanor Avenue, Sunnyvale, California 94086 (408) 732-2710



Copy No. _____
25-00000-14 REVISION B

ATAC-16M Principles of Operation

The ATAC-16M is a microprocessor-based, multi-channel, real-time data acquisition and control system. It is designed to interface with a variety of analog and digital sensors and actuators. The system is capable of sampling data at rates up to 100,000 samples per second and can store up to 1,000,000 samples of data. It is also capable of performing real-time processing of the data, such as averaging, filtering, and integration. The ATAC-16M is controlled by a host computer, which can be a personal computer or a dedicated microcomputer. The host computer sends commands to the ATAC-16M to start and stop data acquisition, and to retrieve the data. The ATAC-16M also sends status information back to the host computer, such as the number of samples acquired and the current status of the system.

THE FOLLOWING INFORMATION IS FOR YOUR INFORMATION ONLY.

This document contains information that is classified as "Confidential" under the provisions of the Arms Control and Disarmament Act, Public Law 90-381, as amended. It is to be controlled, stored, handled, transmitted, and disposed of in accordance with the provisions of that Act and the regulations promulgated thereunder. It is to be released only to those personnel who have been authorized to receive it.

ATAC-16M

Applied Technology

10000 W. 10th Avenue, Suite 100

10000 W. 10th Avenue, Suite 100
Denver, Colorado 80202

INSTRUCTION INDEX

Addition	ADD
Logical Product	AND
Branch and Link	BAL
Branch on Condition	BRC
Compare Between Limits	CBL
Clear Interrupts	CINT
Compare Masked Equality	CME
Compare Address	CMPA
Compare Logical	CMPL
Compare Signed	CMPS
Clear Memory Semaphore	CMS
Double Precision Addition	DADD
Division	DIV
Disable Interrupts	DSI
Double Precision Subtraction	DSUB
Enable Interrupts	ENI
Floating Point Addition	FADD
Floating Point Division	FDIV
Fix (Convert to Fixed Point)	FIX
Float (Convert to Floating Point)	FLT
Floating Point Multiplication	FMUL
Floating Point Subtraction	FSUB
Halt	HALT
Increment and Branch Negative	IBN



INSTRUCTION INDEX

Inclusive OR	IOR
Load Arithmetic (2's) Complement	LAC
Load Lower Byte	LDLB
Load-Register	LDR
Load Multiple	LDRM
Load Status	LDST
Load Upper Byte	LDUB
Load Logical (1's) Complement	LLC
Multiplication	MUL
No Operation	NOP
Return Multiple Register	RET
Register Input	RIN
Register Output	ROUT
Search Lower Byte	SCHL
Search Word	SCHW
Shift Double Register	SHD
Shift Single Register	SHS
ATAC Simulator Control	SIM
Set Memory Semaphore	SMS
Stack Multiple Register	STK
Store	STR
Store Multiple	STRM
Subtraction	SUB
Swap Interrupt Mask	SWIM
Initiate TRAP Interrupt	TRAP
Exchange Registers	XCH
Exchange Byte, Register	XCHB
Exchange Multiple	XCHM
Execute Modified Instruction	XMDI
Exclusive-OR	XOR

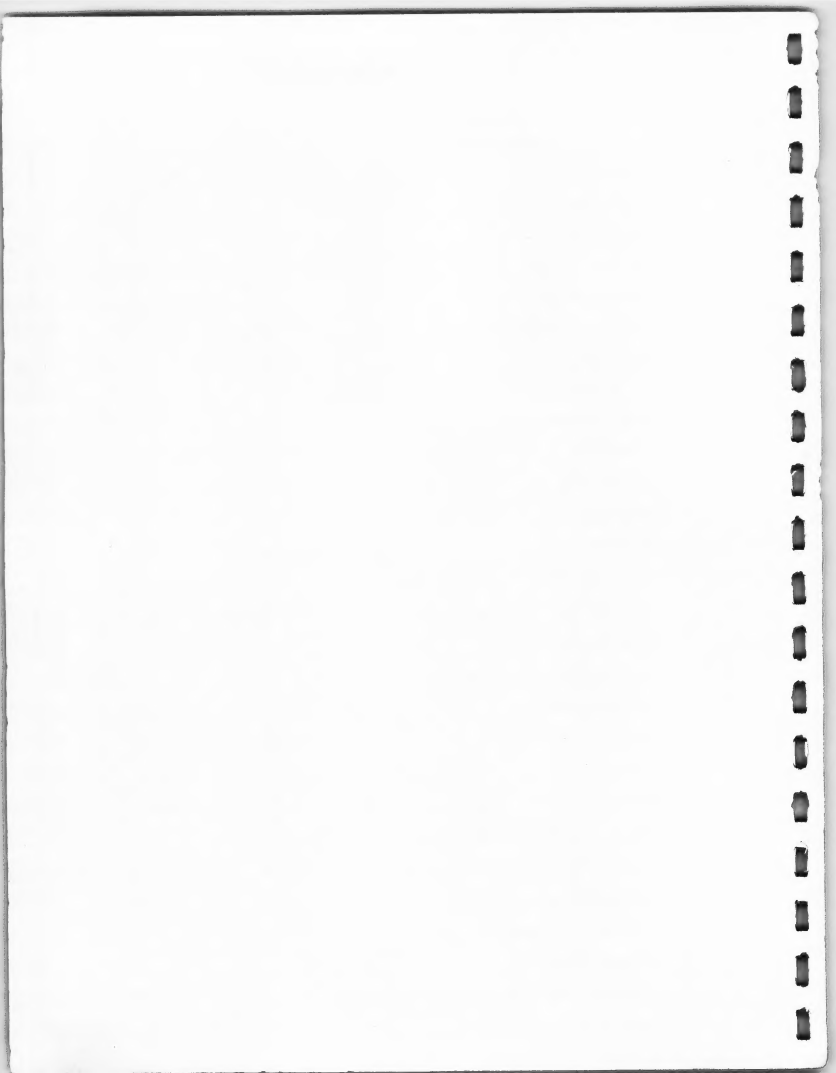


TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
1.0 INTRODUCTION	1-1
1.1 ATAC-16M Technical Specifications	1-2
1.1.1 General Organization	1-2
1.1.2 Word Size	1-2
1.1.3 General Registers	1-2
1.1.4 Addressing Modes	1-2
1.1.5 Memory Interface	1-2
1.1.6 Instruction Set	1-5
1.1.7 Execution Times	1-5
1.1.8 Programmed I/O	1-5
1.1.9 Direct Memory Access	1-5
1.1.10 Interface Signal Levels	1-5
1.1.11 Interrupts	1-7
1.1.12 Special Features	1-7
1.1.13 Physical Characteristics	1-7
1.1.14 Software - Offline	1-8
1.1.15 Software - Online	1-8
1.2 Applicable Documentation	1-9
2.0 ATAC-16M COMPUTER ARCHITECTURE	2-1
2.1 General Register File - ALU	2-1
2.2 Condition Code Register (CCR) Status	2-3
2.3 Register File Address Logic	2-3
2.4 Utility Register File	2-3
2.5 Microprogrammed Control	2-4
2.6 Memory-I/O Interface Registers	2-5
2.7 Interrupt Control	2-6
2.8 Clock and Timing Logic	2-6
3.0 INPUT-OUTPUT FACILITIES	3-1
3.1 Parallel I/O	3-1
3.2 Interrupt Processing	3-2
4.0 CONTROL PANELS	4-1
4.1 ATAC Mini Control Console	4-1
5.0 PHYSICAL SPECIFICATIONS	5-1
5.1 Reliability	5-1
5.2 Mechanical Configuration	5-2
5.3 Space-Qualified Configuration	5-3

TABLE OF CONTENTS (CONTINUED)

<u>Section</u>	<u>Page</u>
6.0 INSTRUCTION SET	6-1
6.1 Numerical Representation	6-1
6.1.1 Single Precision Fixed Point	6-1
6.1.2 Double Precision Fixed Point	6-1
6.1.3 Floating Point Numbers and Arithmetic	6-2
6.1.4 Floating/Fixed Conversions	6-3
6.1.5 The Condition Code Register	6-3
6.2 Instruction Formats	6-6
6.3 Addressing Modes	6-8
6.4 Instructions	6-11
Addition	6-13
AND	6-21
Branch and Link	6-27
Branch on Condition	6-33
Compare Between Limits	6-39
Clear Interrupts	6-43
Compare Masked Equality	6-45
Compare Address	6-51
Compare Logical	6-57
Compare Signed	6-61
Clear Memory Semaphore	6-67
Double Precision Addition	6-73
Division	6-77
Disable Interrupt Network	6-79
Double Precision Subtraction	6-81
Enable Interrupt Network	6-85
Floating Point Add	6-87
Floating Point Divide	6-89
Fix	6-91
Float	6-93
Floating Point Multiply	6-95
Floating Point Subtract	6-97
Halt	6-99
Increment and Branch Negative	6-101
Inclusive OR	6-107
Load Arithmetic (Two's) Complement	6-113
Load Lower Byte	6-119
Load Register	6-125
Load Multiple	6-133
Load Status	6-137
Load Upper Byte	6-143
Load Logical (One's) Complement	6-149
Multiply	6-155

TABLE OF CONTENTS (CONTINUED)

<u>Section</u>	<u>Page</u>
No Operation	6-161
Return	6-163
Register Input Instruction	6-165
Register Output Instruction	6-167
Search Lower Byte, Every Nth Word	6-169
Search Every Nth Word	6-175
Shift Double Register	6-183
Shift Single Register	6-187
ATAC Simulator Control Instruction	6-191
Set Memory Semaphore	6-193
Stack	6-201
Store	6-205
Store Multiple	6-209
Subtraction	6-213
Swap Interrupt Mask	6-221
Trap	6-223
Exchange Registers	6-225
Exchange Bytes	6-227
Exchange Multiple	6-229
Execute Modified Instruction	6-231
Exclusive OR	6-233
APPENDIX A	Symbolic Condition Branch Entries
	A-1
APPENDIX B	Instruction Set Summary
	B-1
APPENDIX C	ATAC-16M Instruction Execution Times
	C-1
APPENDIX D	Symbols, Notation, and Abbreviations
	D-1
APPENDIX E	Conversion Aids
	E-1
APPENDIX F	Floating Point Numbers - Conversions and Examples
	F-1
APPENDIX G	ATAC-16M Master/Reset Start-Up Sequence
	G-1

LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1-1	ATAC-16M Memory Interface (Read Cycle Timing)	1-3
1-2	ATAC-16M Memory Interface (Write Cycle Timing)	1-4
2-1	ATAC-16M Organization	2-2
3-1	ATAC-16M Parallel Input/Output (PIO) Interface Timing	3-3
3-2	ATAC-16M Interrupt Processing Scheme	3-6
3-3	ATAC-16M Interrupt Processing Flowchart	3-7
3-4	ATAC-16M Program Status Word (PSW)	3-9
4-1	ATAC Mini Control Console	4-2
6-1	Instruction Format/Addressing Modes	6-7

LIST OF TABLES

<u>Table</u>		<u>Page</u>
1-1	Typical ATAC-16M Interface Drive and Load Parameters	1-6
	Table of 'BM' (Branch Mode) Mnemonics	A-3
E-1	Hexadecimal Arithmetic Tables	E-2
E-2	Powers of Two Table	E-3
E-3	Largest Decimal Number Table	E-4

1.0

INTRODUCTION

Through advanced, computer-aided design techniques, Applied Technology's Advanced Computer (ATAC) has been developed as a highly optimized, general-purpose, real-time processor. It uses advanced technology to achieve very high computation speed with minimal physical requirements, and the initial instruction set is optimized for real-time avionic applications.

ATAC-16M is a member of the Applied Technology modular family of computer components which permits the system designer to tailor an ATAC computer to the physical and computational requirement. The ATAC family of computer components consists of the following subsystems:

- Central Processors
- Memory Modules
- High Speed Input/Output Channels
- Input/Output Devices
- Control Consoles
- Applications Support Software

Standard ATAC features include a microprogrammable computer architecture, asynchronous interface to all memory types, expandable basic instruction set, register I/O, and a priority-interrupt structure. Optional features include multiple direct memory access ports, control panel with hardware breakpoint, and I/O interfaces. These features qualify ATAC for application in a wide range of real-time and process-control systems.

This publication describes the ATAC-16M computer principles of operation. It provides a comprehensive overview of the system structure and the instruction repertoire.

A space-qualified version called ATAC-16MS, which is functionally equivalent but tailored for conduction cooling, is described in the ATAC-16MS Processor Specification (20-059402). The instruction level operation of the ATAC-16MS is described in this document.

1.1 ATAC-16M TECHNICAL SPECIFICATIONS

1.1.1 General Organization

The ATAC-16M is a general-purpose, binary, parallel 2's complement, integer fixed point and fractional floating point, general register, microprogrammed, fully overlapped computer.

1.1.2 Word Size

Instructions:	16, 32-bit
Data fixed point:	8, 16, 32-bit
Data floating point:	32-bit 24/8 format

1.1.3 General Registers

The ATAC-16M provides sixteen general-purpose 16-bit registers, each useable as an accumulator, stack pointer, index register or indirect pointer.

1.1.4 Addressing Modes

Eight addressing modes are provided: Register, Immediate, Direct, Direct Indexed, Register Indexed, Immediate Short, Direct Relative, Program Counter Relative.

1.1.5 Memory Interface

The asynchronous, fully buffered interface accommodates a wide variety of memory modules (such as PROM, RAM, or core) in the same system, but with different access times. The ATAC-16M processor interfaces directly to standard ATAC memory boards without the need for additional timing or control logic. Up to 65,536 words of memory can be directly addressed..

Memory interface timing is shown in figures 1-1 and 1-2.

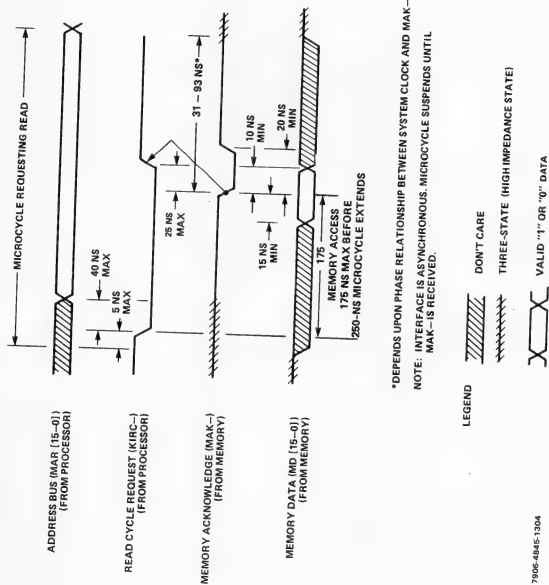
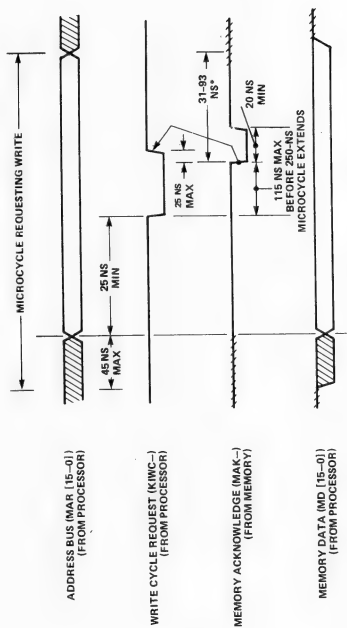
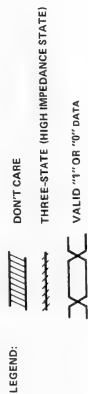


Figure 1-1. ATAC-16M Memory Interface (Read Cycle Timing)



*DEPENDS UPON PHASE RELATION BETWEEN SYSTEM CLOCK AND MAK-
NOTE: INTERFACE IS ASYNCHRONOUS, MICROCYCLE SUSPENDS UNTIL MAK- IS RECEIVED



7906 4845 1305

Figure 1-2. ATAC-16M Memory Interface (Write Cycle Timing)

1.1.6 Instruction Set

One hundred thirty-one instructions are optimized for real-time avionics applications and high-speed arithmetics. Other sets can be easily microprogrammed to suit applications.

1.1.7 Execution Times

Instruction times are based on an integral 16-MHz system clock, yielding a 250-nanosecond microcycle, and are dependent upon the memory system used. Complete timing data for each instruction for three memory systems is given in Appendix C. Some typical times are given below:

	Add/ Subtract	Multiply	Divide	Indexed Load
Fixed Point (μ sec):	0.25	5.5	11.25	0.75
Floating Point (μ sec):	5.75	16.0	28.5	1.25

1.1.8 Programmed I/O

Features include a sixteen-bit, fully buffered, parallel, asynchronous I/O bus; 65,536 addressable I/O devices; and 500,000 word transfers per second. Special-purpose and general purpose I/O boards are available.

1.1.9 Direct Memory Access

The ATAC-16M CPU interfaces with a multichannel high speed DMA controller, and places no throughput constraints on the DMA process.

1.1.10 Interface Signal Levels

All signal levels are TTL compatible. Drive and loading are as specified in table 1-1.

Table 1-1. Typical ATAC-16M Interface Drive and Load Parameters

	<u>Output Drive</u>	<u>Input Load</u>
1) Memory/I/O Address Bus 0-15	48 mA Sink (0) 12 mA Source (1)	-
2) Memory/I/O Data Out Bus 0-15	20 mA Sink (0) 2 mA Source (1)	-
3) Memory/I/O Data In Bus 0-15	-	20 μ A (1) -0.4 mA (0)
4) Interrupt Input Lines 0-7	-	40 μ A (1) -0.72 mA (0)
5) I/O Device Demand	16 mA Sink (0) 0.8 mA Source (1)	-
6) I/O Device Acknowledge	-	40 μ A (1) -16 mA (0)
7) Memory Read (Write) Request	16 mA Sink (0) 0.9 mA Source (1)	-
8) Memory Acknowledge	-	40 μ A (1) -16 mA (0)

Data and address buses from the processor are driven by tri-state devices and are provided with enable signals which can be externally controlled. Memory and I/O acknowledge signals to the processor are terminated internally in order to accommodate multiple tristate or open collector sources.

1.1.11 Interrupts

There are eight priority interrupt levels (expandable to 64). Interrupt system features include automatic hardware vectoring and linkage, individual mask capability, less than 4-microsecond interrupt service micro-code response, and in-process interrupt capability on certain long instructions (e.g., Search).

1.1.12 Special Features

- High performance, real-time emulation capability
- Built-in control signals for multiprocessor/multicomputer applications
- Mating with Extended Arithmetic Processor for high-speed floating point and double precision (32-bit) operations
- Compatibility with standard TTY, RS-232 and MIL-STD-1553A interfaces

1.1.13 Physical Characteristics

a. Avionics

Environment: MIL-E-5400, class II
 Size: Single PC board, 6 by 8 inches
 Power: 21 watts (single supply, +5 volts)

b. Space

Environment: Vacuum Conduction Cooled
 Size: 6 by 11.5 inches
 Power: 20.2 watts (5-volt single supply)

1.1.14

Software - Offline (Support Software)

- Transportable FORTRAN-based instruction-level assembler
- Transportable FORTRAN-based relocating linking loader with provisions for code assignment to a variety of memory types and mixes
- Instruction-level simulator with automatic program timing, instruction frequency count, and complete trace and snapshot memory dump options
- FORTRAN and HAL-S compilers
- Compiler-level systems programming language designed for real-time processing and systems control
- BASIC interpreter
- Microcode assembler and complete microcode simulator to permit evaluation and development of specialized instructions
- Programming support system with monitor to permit ease of utility software extension
- Programming support software package operating on local or remote IBM 360/370; Assembler/Loader operating on any 16-bit mini-computer which supports FORTRAN IV.

1.1.15

Software - Online

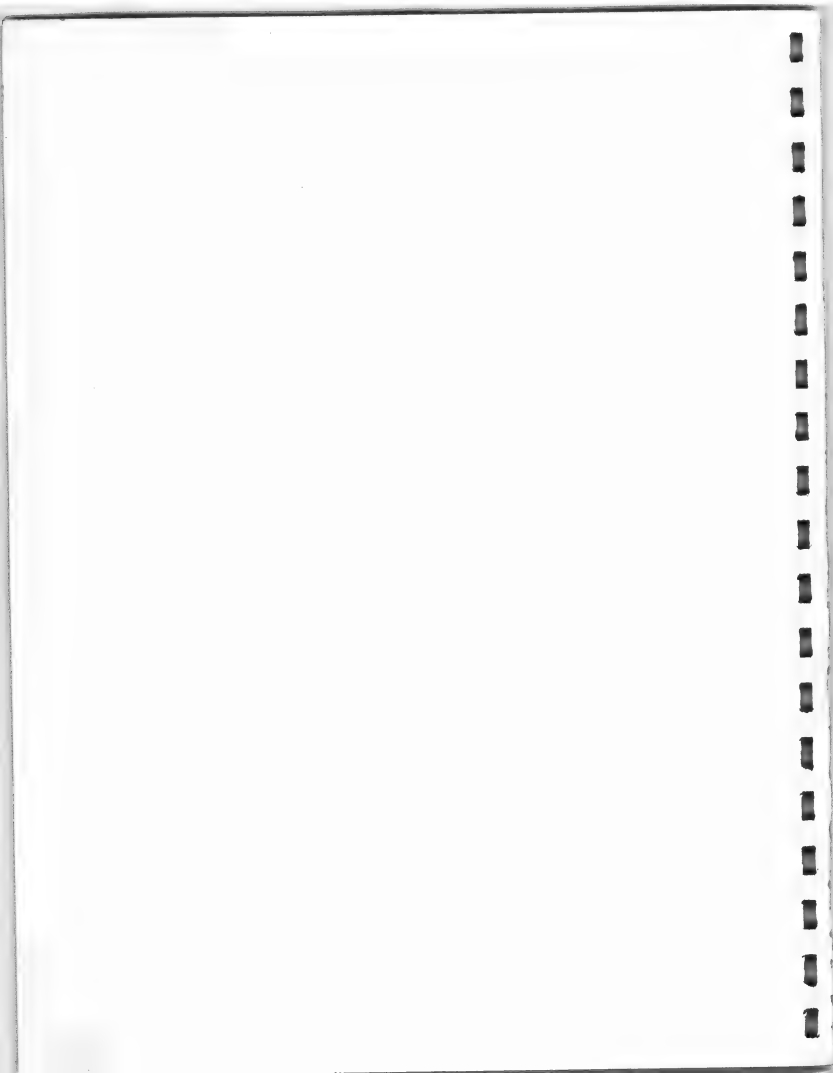
- Diagnostic and test software for instructions, interrupts, I/O and memory

- Debug package
- Self assembler, loader, and operating system

1.2

APPLICABLE DOCUMENTATION

All ATAC computer documentation items are listed in Applied Technology Document No. 52-056128 entitled "ATAC Computer Documentation".



2.0

ATAC-16M COMPUTER ARCHITECTURE

The ATAC-16M computer architecture consists of the following sections which have been organized to meet the design constraints of high computation speed with minimum physical requirements:

- General Register File-ALU
- Condition Code Register Status
- Register File Address Logic
- Utility Register File
- Microprogrammed Control
- Memory-I/O Interface Registers
- Interrupt Control
- Clock and Timing Logic

Each section is discussed individually in the following narrative. Figure 2-1 is a block diagram showing the relationships of the various modules.

2.1

GENERAL REGISTER FILE-ALU

At the heart of the ATAC-16M is the bit-slice 2901 microprocessor. Four of these devices are concatenated together to form the RALU section, which contains a 16-bit, 16-word general register file, an arithmetic logic unit (ALU) (capable of 32 logical and 24 arithmetic operating modes on two 16-bit variables), a 16-bit Q extension register (used for multiply/divide, shift, double precision, and auxiliary storage) and shift logic, permitting the output of the ALU to be shifted (in combination with the Q register, if desired) one bit at a time left or right.

The ALU, which uses carry look-ahead for improved speed, is supplied with two operands from the five input operand multiplexer. This includes a logical 'zero' operand for simply passing data through the ALU to the output Y-bus or internal shift logic.

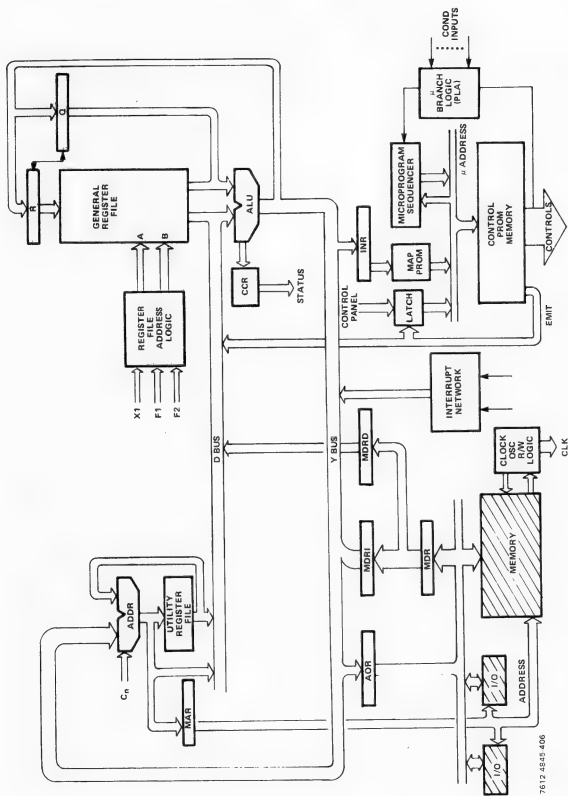


Figure 2-1. ATAC-16M Organization

The 2901 microprocessor is controlled by a nine-bit instruction word from ATAC-16M's control memory. This word is broken into three 3-bit fields internal to the microprocessor to control ALU operand source, ALU operation, and ALU output destination (which also includes shift operations).

2.2 CONDITION CODE REGISTER (CCR) STATUS

The ALU of the 2901 microprocessor supplies an overflow bit, carry bit, sign bit, and output equal zero bit, which are formed during arithmetic and logical operations performed by the ALU. This processor status information is saved in a 4-bit condition code register during certain of these ALU functions, under microprogram instruction control. The resulting CCR code may be tested in the microprogram control section for conditional branching on positive, negative, greater than, less than, zero, or overflow, or any combination of these conditions. The CCR also may source the Y-bus, and in turn receive inputs from this bus, for use in saving and returning process status during and after interrupt processing.

2.3 REGISTER FILE ADDRESS LOGIC

The general register file is supplied with two register addresses (A and B) from the register file address logic, in the form of a 4-bit code for each address. These may be the F1 field or F2 field of the instruction word for the A address, and the F2 field or X1 field of the instruction word for the B address. This logic also provides sign extension of the X1, F1 literal field of immediate instructions and movement to the microprocessor's input D-bus.

2.4 UTILITY REGISTER FILE

The utility register file consists of four, 4-bit-slice LSI devices concatenated together to form a 16-bit, 3-register internal file, a two-port output (consisting of a 16-bit output register on one port, and a 2:1

multiplexer on the other) and a 16-bit input adder. Two of the three internal registers are used for general scratch registers by the microprogram control section, while the third is used for the program counter (PC). The output register is used as the memory address register (MAR).

The input to the utility register file is first passed through the adder, whose one input is the Y-bus (or the Y-bus with upper and lower bytes swapped) and whose other input is from one of the three internal utility registers. This structure permits incrementing of the PC (via carry-in), forming effective addresses by combining the PC with Y-bus data, or simply provides a path for passing data from the ATAC-16M's Y-bus to the D-bus.

2.5

MICROPROGRAMMED CONTROL

The microprogrammed control section of ATAC-16M consists of all the logic required to control the processor so that the selected instruction set can be implemented. Actually a small computer within a computer, the control section consists of a PROM memory with latched output; this memory contains the micro-instruction set required to perform control and sequencing operations of all logic within the ATAC-16M to perform any instruction in the processor's repertoire. Broken into 20 field groupings, these memory bits control the 2901 microprocessor, the CCR, the register file address logic, utility register file section, all memory-I/O interface registers, the interrupt network section and the clock and timing logic.

One of four sources provides the address to the control memory: the MAP PROM, microprogram sequencer, feedback from the control PROM, or the control panel. The MAP PROM uses the current instruction's operation code field (held in the INR register) to form a rearranged address for the control memory, allowing up to 256 operation codes. The microprogram sequencer provides incrementing by one, as well as subroutine jumps and return linkage capability for the microprogram address. The control memory feedback path (EMIT) provides direct microcode address jumping ability.

The correct source for the next microprogram address is selected by the microcode branch conditional logic (an FPLA), based on the condition selected by the control memory and conditional inputs both from internal (such as processor status from the CCR) and external (such as control panel request) sources.

2.6

MEMORY-I/O INTERFACE REGISTERS

There are three input and two output 16-bit registers associated with main memory and input/output device interface. The memory data register (MDR) is an asynchronous buffer latch between ATAC-16M and external asynchronous memory and I/O devices. Data clocked into this register is not used until the end of the current micro-instruction, decoupling main memory or I/O devices and internal processor timing. If the input data is an instruction word from main memory, it is passed to the memory data register for instructions (MDRI), where it is held for next instruction decoding by the control section's MAP PROM. If the input word from memory or the I/O device is data, it passes to the memory data register for data (MDRD), whose output sources the 2901 microprocessor's input data bus (D-bus). The double rank of input registers forms the pipelined architecture of the ATAC-16M, which permits instruction overlap that results in high speed instruction execution.

The memory address register (MAR) within the utility register file is buffered and sent off the processor card as either main memory or I/O device address. Neither of the two share the same address space, and thus each has 65,536 possible addresses. The arithmetic output register (AOR) holds either main memory or I/O device destined output data. The three-state output of this register may be tied directly to the MDR inputs, to form a bi-directional memory and I/O device data bus, or may be left unattached to provide separate unidirectional input and output data buses for the ATAC-16M.

Microbit KIOD - is active for I/O transfers only.

2.7 INTERRUPT CONTROL

The priority interrupt network of the ATAC-16M consists of a single LSI circuit that provides eight fully prioritized, vectored, automatic interrupt levels. Pinouts are provided on the card I/O connector for external expansion up to 64 levels, in increments of eight levels. The interrupt network contains a rank of flip-flops which capture incoming interrupt signals and an interrupt mask register which, under program control, allows an interrupt to be processed or ignored.

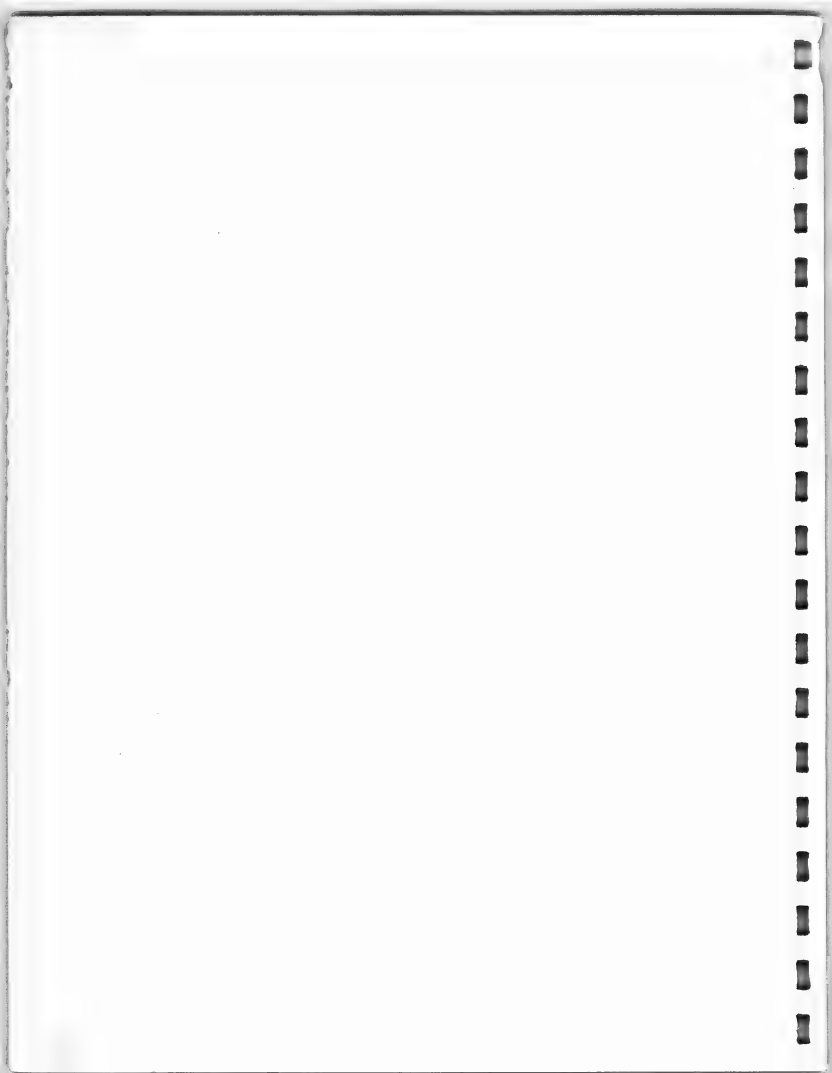
The interrupt network also possesses a status register that holds the encoded level of the currently highest priority interrupt level enabled, and uses this knowledge to provide interrupt level nesting (interrupt within an interrupt) for up to the full eight levels. Under microcode control, interrupts are vectored to the appropriate service routine, without the need for software overhead. Interrupt microcode response is 3.25 microseconds, plus memory response time (if any), plus instruction latency.

The ATAC-16M also provides the capability of direct microcode interrupt service, skirting the normal interrupt service overhead for high speed interrupt (less than 1 microsecond) response.

2.8 CLOCK AND TIMING LOGIC

The ATAC-16M contains a 16-MHz crystal-controlled clock oscillator, whose output is counted down to 4 MHz to produce the basic 250-nanosecond microcycle clock, used for clocking all internal registers on the processor. The timing logic contains a built-in synchronizer which provides the 250-nanosecond clock with a halt and restart feature that permits ATAC-16M to operate with memory and I/O devices of varying speeds, or to operate with an externally supplied oscillator, if desired.

The timing logic generates the read and write requests to main memory or I/O devices under microprogram control, and accepts acknowledgement when data transfer is complete. In this manner the microcycle is either the normal 250 nanoseconds, for fast memory, or extended for slow access memory or I/O devices during a processor read or write microcycle.



3.0 INPUT-OUTPUT FACILITIES

3.1 PARALLEL I/O

ATAC-16M's parallel I/O consists of a set of resources which can be programmed, using microcode, to provide a spectrum of I/O capability. The intent of this I/O structure is to provide ATAC-16M with the facility to interface to a variety of devices using a minimum of hardware. Synchronous or asynchronous, parallel or serial, multiword or single word transfers can be accommodated exclusively or in combination by ATAC-16M.

The primary resources of ATAC's I/O are:

- A 16-bit, bidirectional, 3-state bus for asynchronous data transfer (which may also be configured as separate unidirectional input and output data buses)
- A 16-bit address word
- A control line for strobing address and data to I/O devices ?
;
- An input acknowledge line for I/O devices to signal their response to instructions or data **PIDACK**

Standard programmed data input/output occurs between the I/O device and the register file. Any of the 16 registers in the file can output to or receive input from an I/O device. The 16-bit bidirectional bus is a buffered, TTL, 3-state bus easily interfaced to a wide range of peripherals. The 16-bit address word can be used to define sources, destinations, and functions during parallel I/O operations.

A control line referenced as ~~K~~IOD (I/O demand) identifies the I/O bus status to the peripherals. An active IOD implies the I/O address is valid and the I/O buses are now ready to accommodate an I/O operation. An asynchronous peripheral normally executes the I/O instruction and signals the completion of the instruction by responding with an acknowledge pulse.* If the peripheral does not respond with an acknowledge pulse within 64 microseconds, the ATAC-16M automatically terminates the operation, and sets overflow as an indication that the I/O operation has been timed out.

*PIODACK

Figure 3-1 is a timing diagram for PIO interface timing.

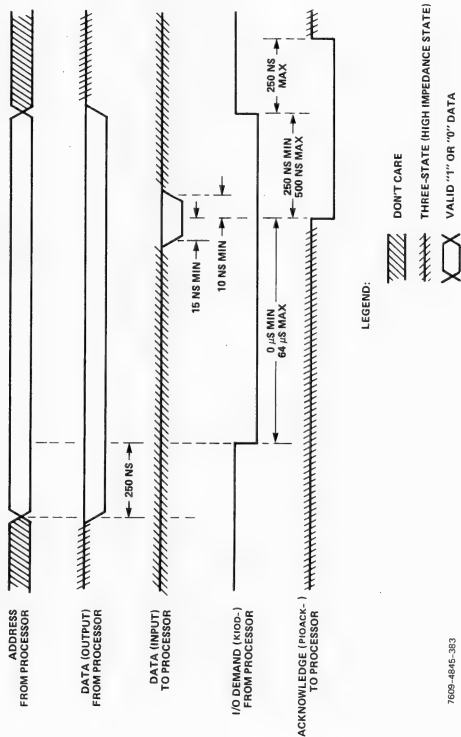
3.2

INTERRUPT PROCESSING

The priority interrupt network of the ATAC-16M provides eight fully prioritized and maskable interrupt levels on the processor card. The interrupt network is expandable to 64 levels in increments of 8. The processor provides full interrupt priority processing without software overhead.

Incoming signals requesting an interrupt of the ATAC-16M are asynchronously 'latched in' and remembered whenever they are received.* With the exceptions described below, the processor examines all interrupts awaiting service ('pending interrupts') after the execution of each instruction. If there is an 'armed' interrupt (see below) and if the highest priority 'armed' interrupt is of higher priority than the interrupt level currently being serviced (if any), then the processor will pass control to the special interrupt service routine corresponding to the interrupt priority level just recognized.

*Interrupt signals to the ATAC-16M are active low pulses of duration greater than 0.03 microseconds and less than 1.0 microsecond.



7609-4845-383

Figure 3-1. ATAC-16K Parallel Input/Output (PIO) Interface Timing

Interrupts to the ATAC-16M are automatically prioritized by the use of the interrupt priority register. No interrupt of a lower priority than the priority level stored in the interrupt priority register may interrupt the ATAC-16M. The interrupt priority register is automatically updated each time an interrupt is processed. Prioritization is automatic and thus totally transparent to the programmer.

The programmer may at any time set an interrupt mask using the SWIM (Swap Interrupt Mask) instruction. Each bit in the interrupt mask corresponds to an individual interrupt level. Bit 0 (the least significant bit) corresponds to interrupt level 0 (the highest priority interrupt). A '1' in the bit location of the interrupt mask corresponding to a given interrupt level allows an interrupt on that interrupt level to be recognized by the processor, subject to the condition that that interrupt level is of the same priority or a higher priority than the level stored in the interrupt priority register. The particular interrupt is thereby 'armed' or 'enabled'. Conversely, a '0' in a bit of the interrupt mask corresponding to a given interrupt level 'disarms' or 'disables' that interrupt level.

All interrupts may be disabled without disturbing the interrupt priority register or the interrupt mask by using the DSI (Disable Interrupts) instruction. All interrupts are inhibited after execution of the DSI instruction until the ENI (Enable Interrupts) instruction is invoked.

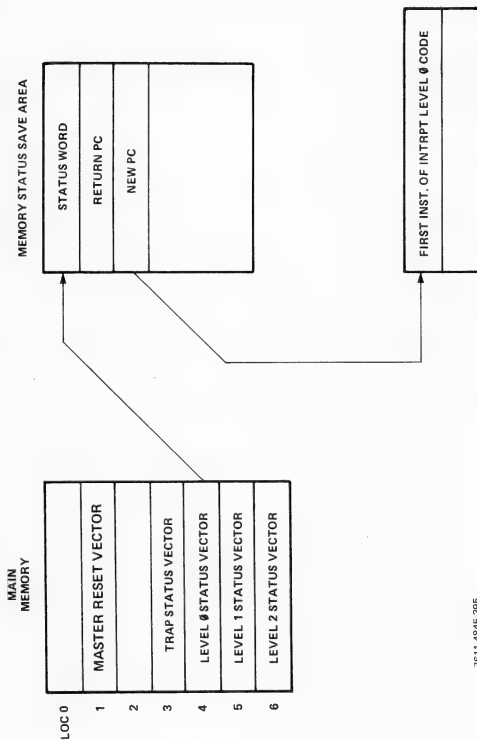
No matter what the state of the interrupt priority register and the interrupt mask, any incoming interrupt requests will be latched in and 'remembered' to be processed later on. Even if the interrupts have been disabled using the DSI instruction, they will continue to be latched in. An interrupt remains 'pending' until it is either serviced or cleared using the CINT instruction.

The CINT (Clear Interrupts) instruction allows the programmer to selectively clear pending interrupts regardless of the current interrupt processing level. The CINT instruction uses a mask (in the same format as the 'interrupt mask') which is set up in one of the 16 general purpose registers. A '1' in this mask clears the interrupt level corresponding to the bit position of the '1'. Bit position 0 (least significant bit) corresponds to interrupt level 0 (highest priority).

After the ATAC-16M has been master reset, the interrupt priority register will contain the lowest priority level possible, given the particular interrupt configuration of the particular ATAC-16M computer. The interrupt mask register will be set to all '1's so that all interrupts will be enabled. However, the interrupt chip will be disabled as though the DSI instruction had been executed. Consequently, no interrupts will be recognized until software issues an ENI instruction. Hence, initialization of an executive program may be accomplished without being interrupted. (See also appendix G for Master Reset/Power On Sequence.)

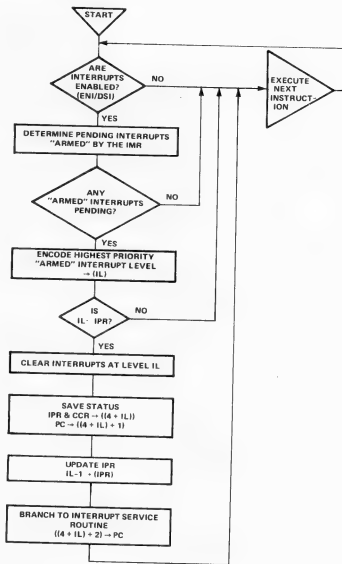
The normal interrupt servicing scheme is diagrammed in figures 3-2 and 3-3, and is described below:

<u>Step</u>	<u>Operation</u>
1	The PSW is read from the CCR and the IPR and is stored in a temporary register.
2-3	The active interrupt level is determined. The new IPR value (active interrupt level minus 1) is computed and the IPR is updated.
4	The active interrupt is cleared.
5	The address of the pointer to the three-word status save area associated with the active interrupt level is computed by adding 4 to the active interrupt level.
6-7	The status save area pointer is read from memory location (interrupt level + 4).



7611 4845-395

Figure 3-2. ATAC-16M Interrupt Processing Scheme



NOTES:

- (1) INTERRUPTS ARE LATCHED (REMEMBERED) IRRESPECTIVE OF THE STATES OF THE INTERRUPT AND PROCESSOR HARDWARE; I.e. INTERRUPTS ENABLED OR DISABLED, IMR SET OR CLEARED.
- (2) INTERRUPTS REMAIN LATCHED, 'PENDING' SERVICE UNTIL THEY ARE CLEARED EITHER BY THE EXECUTION OF THE APPROPRIATE INTERRUPT SERVICE ROUTINE, OR BY THE INVOCATION OF THE CINT INSTRUCTION.
- (3) ANY INTERRUPT PULSES REQUESTING AN ADDITIONAL INTERRUPT AT A LEVEL WHICH IS ALREADY 'PENDING' WILL BE IGNORED UNTIL THE 'PENDING' INTERRUPT IS SERVICED.

KEY:

(IL) INDICATES THE CONTENTS OF THE MEMORY LOCATION ADDRESSED BY IL (INTERRUPT LEVEL).

((4 + IL)) INDICATES THE CONTENTS OF THE MEMORY LOCATION ADDRESSED BY THE CONTENTS OF MEMORY LOCATION 4 + IL.

7712-4845 701

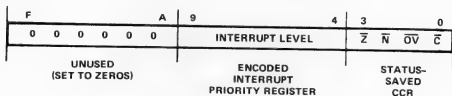
Figure 3-3. ATAC-16M Interrupt Processing Flowchart

<u>Step</u>	<u>Operation</u>
8	The program status word is read from its temporary storage register and is written to the first word of the status save area.
9	The address of the instruction interrupted prior to execution is stored in the next location of the status save area.
10	The beginning address of the appropriate interrupt service routine is read from the last word in the status save area.
11-13	The processor performs an instruction fetch and begins execution of the interrupt service routine.

The ATAC-16M program status word (PSW) is shown in figure 3-4. Bits 0 to 3 of the PSW contain the contents of the condition code register at the time of interrupt. As many bits of the PSW as are needed for the interrupt configuration (up to 6 bits) starting with bit 4 are used to store the interrupt priority register from the hardware interrupt network at the time of interrupt. For example, with configurations providing 16 levels of interrupts, the interrupt priority register is stored in bits 4 through 7. The remaining high order bits are filled with zeros.

The condition code register contains four bits of ALU status information: a zero ALU result indicator, a negative ALU result indicator, an arithmetic overflow indicator, and the ALU carry out. The relationship between this hardware CCR, the version of the CCR relevant to the programmer and the CCR as saved in the program status word is shown in figure 3-4.

The interrupt level read from the interrupt priority register (in step 1 above) and saved as part of the program status word is one less than the interrupt priority level (IL - 1) at which the processor was running before the new interrupt occurred. The IPR value stored in steps 2-3 above is the new active interrupt priority level, less 1. The interrupt level is independent of any interrupt mask value.



SOFTWARE TESTABLE CCR	HARDWARE CCR	STATUS- SAVED CCR
OV N Z P	Z N OV C	Z N OV C
0 0 0 1	0 0 0 X	1 1 1 \bar{X}
0 0 1 0	1 0 0 X	0 1 1 \bar{X}
0 1 0 0	0 1 0 X	1 0 1 \bar{X}
1 0 0 1	0 0 1 X	1 1 0 \bar{X}
1 0 1 0	1 0 1 X	0 1 0 \bar{X}
1 1 0 0	0 1 1 X	1 0 0 \bar{X}

7906 4845 1313

Figure 3-4. ATAC-16M Program Status Word (PSU)

For example, suppose the processor is currently executing the service routine for interrupt level 3, then the interrupt priority register will contain a 2. Now, suppose that while the processor is still executing the service routine for interrupt level 3, an interrupt of level 1 occurs and that this interrupt level is armed. The interrupt microcode will then save a 2 as the interrupt level stored as part of the PSW in the status save area reserved for interrupt level 1. The interrupt microcode will then update the interrupt priority register to a 0.

As another example, suppose that an ATAC-16M configuration allows only 8 interrupt levels, that the ATAC-16M is operating in the main program (i.e., the ATAC is not executing an interrupt routine), that the hardware CCR has just been set to 0010 and the next to the highest priority interrupt is armed and becomes active. The following steps are taken:

<u>Step</u>	<u>Operation</u>
1	The interrupt priority register and the CCR are read and temporarily saved. The interrupt priority register is updated with a 0.
2	The processor reads the location of the three-word status save area for interrupt level 1 from memory location 5.
3	The following program status word is formed and saved in the first word of the status save area:

0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1
Priority Register									CCR						

4	The address of the instruction which the processor was about to execute before the interrupt was recognized is written to the second word of the status save area.
---	--

Step

5

Operation

Finally, the starting address of the interrupt service routine for interrupt level 1 is read from the third word of the program save area. This address is loaded into the program counter of the ATAC-16M, and the processor begins executing the interrupt service routine.

Now suppose that the highest priority device requests an interrupt (the only device which can be recognized), that this interrupt is enabled, and that the hardware CCR has now just been set to 0100. The following occurs:

Step

1

Operation

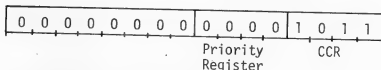
The interrupt priority register and the CCR are read and temporarily saved. Since the interrupt priority register cannot store a higher priority level than 0, hardware internal to the ATAC-16M disables the interrupt chip so that no more interrupts may occur until the execution of the LDST (Load Status) takes place. While the interrupt chip is disabled, incoming interrupts are latched in.

2

The processor reads the location of the three-word status save area for interrupt level 0 from memory location 4.

3

The following program status word is formed and saved in the first word of the status save area:



4

The address of instruction which was about to be executed when the interrupt occurred is written to the second word of the status save area.

StepOperation

5

Finally, the starting address of the interrupt service routine for interrupt level 0 is read from the third word of the program save area. This address is loaded into the program counter of the ATAC-16M, and the processor begins executing the interrupt service routine.

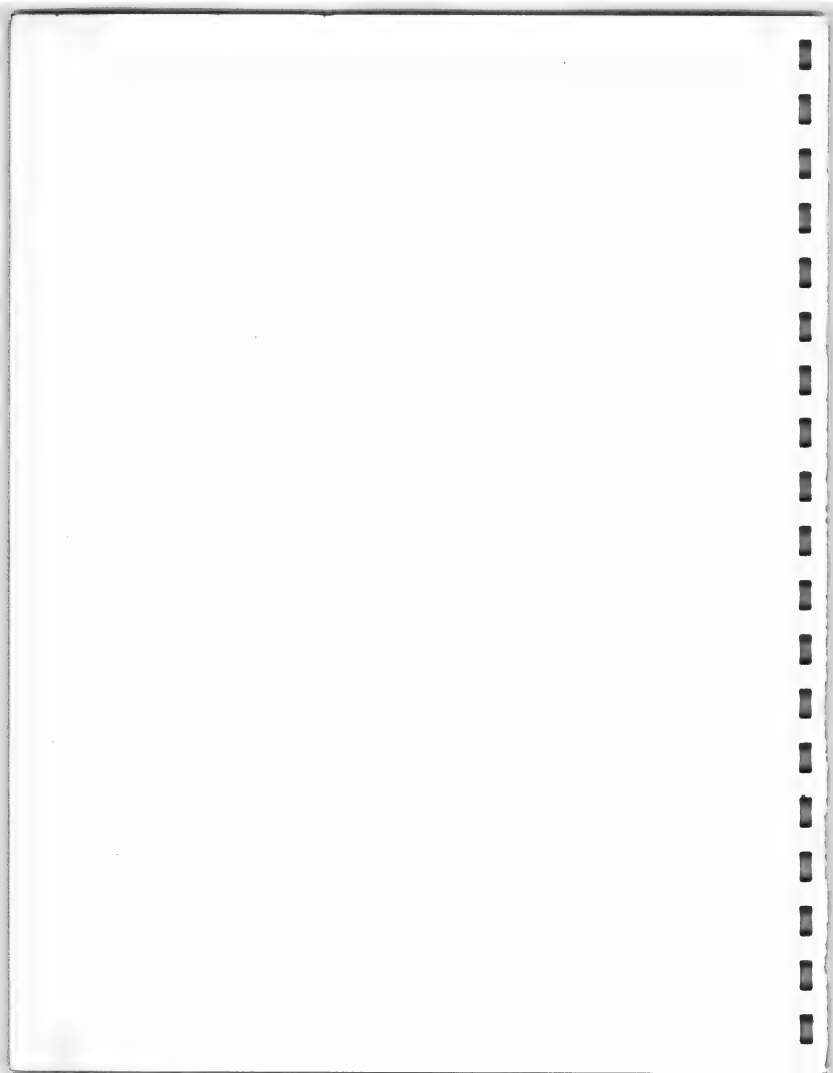
Interrupt service routines are terminated with the LDST (Load Status) instruction. The LDST instruction interprets the direct or direct indexed operand provided to determine the beginning address of the appropriate status save area. The program status word is read and the CCR and IPR are restored to the values they held before the current interrupt was honored. The address of the instruction which was about to be executed before the current interrupt occurred is loaded into the program counter of the ATAC-16M, and control is passed to this point. The LDST instruction is the only means the programmer has to restore the CCR and the interrupt priority register to their previous states.

The TRAP instruction causes the interrupt logic to be executed as though it were an instruction. However, the TRAP instruction leaves the value of the IPR unchanged. The status save pointer for the TRAP instruction is read from memory location 3.

With only the exceptions listed below, interrupts can be recognized between the execution of any two instructions. All interrupts are disabled between each of the following instructions and the instruction immediately following it:

ENI	Enable Interrupts
DSI	Disable Interrupts (de facto)
LDST	Load Status
TRAP	Trap
XMDI	Execute Modified Instruction

Once an interrupt has been recognized, the first instruction of the appropriate interrupt service routine will be executed regardless of any new incoming interrupts of higher priority. Hence, if the DSI instruction is the first instruction of an interrupt service routine, the service routine is uninterruptable by any interrupt priority level; interrupts will be allowed only after the ENI instruction is encountered.



4.0

CONTROL PANELS

ATAC-16M is supported by a control panel device designated the "ATAC minicontrol console". It may be used with ATAC computer systems for software development and hardware checkout as described in section 4.1. With the control panel disconnected, the ATAC-16M processor is automatically initialized by the internal microprogram.

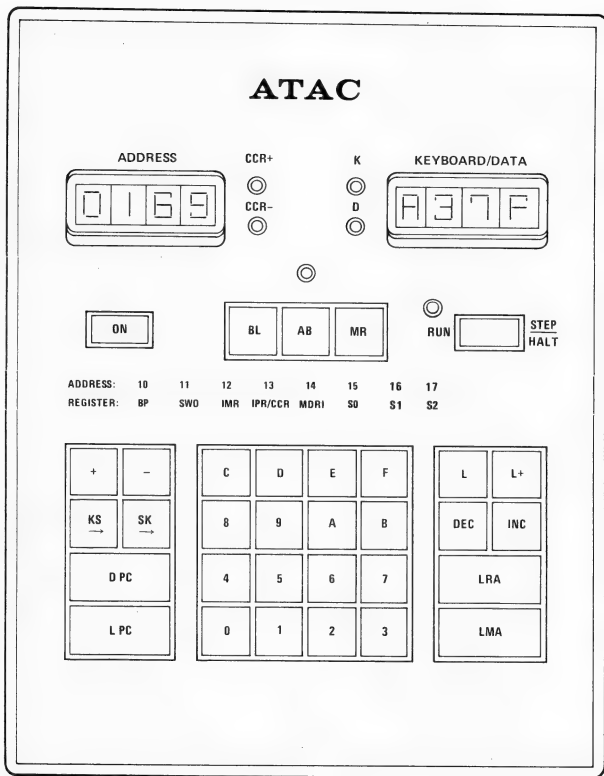
4.1

ATAC MINI CONTROL CONSOLE

To aid development of software and to assist in system integration and testing, the mini control console with comprehensive control functions is available. Figure 4-1 illustrates the control panel layout.

Mini control panel features are:

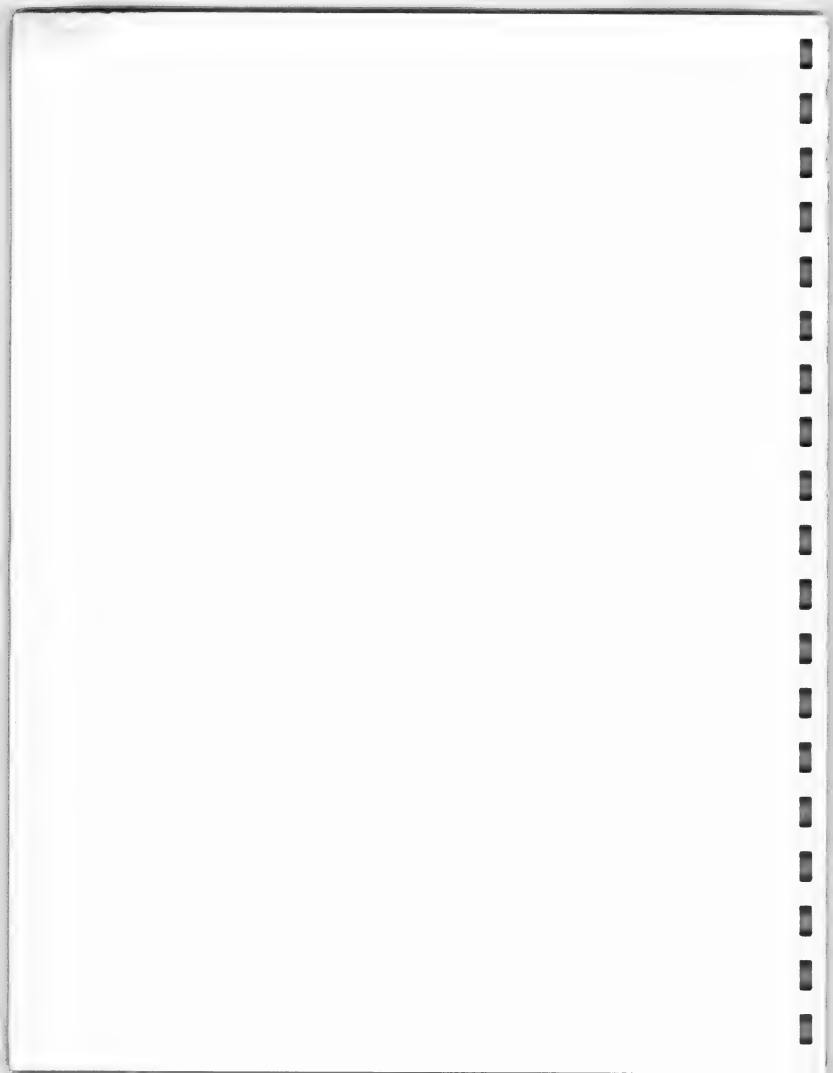
- Hexadecimal input/output KEYBOARD, and hexadecimal READOUT register for entering addresses, counts, or data
- Built-in real-time breakpoint to provide an automatic program halt at specified memory address register values
- SELECT SWITCHES to permit inspection and modification of all registers
- Memory addressing and data control to permit transfer of keyboard contents into any portion of selected memory
- INCREMENT/DECREMENT MEMORY ADDRESS controls for main memory reference
- Programmable status word displayed on panel during computer operation



7505 4845 405

Figure 4-1. ATAC Mini Control Console

- RUN/HALT switch to permit ATAC to commence automatic sequencing
- SINGLE INSTRUCTION control to permit manual control over hardware to trace machine cycles on an instruction-by-instruction basis
- Hexadecimal address addition and subtraction option
- Condition code register and memory fetch phase displayed
- Bootstrap load control to permit operator to invoke the ATAC absolute loader
- Built-in teletype interface



5.0 PHYSICAL SPECIFICATIONS

5.1 RELIABILITY

The high performance and long-life reliability of ATAC-16M is the result of ATI's unique fifth generation design, material, manufacturing and testing techniques. Beginning at the critical design state with stringent "worst case" criteria, development engineers created a system to meet the environmental conditions as well as satisfy performance requirements. Mass production, state-of-the-art techniques utilize small, medium and large-scale integrated circuits with completely wire-free construction. The computer has no processor wires, dissipates less power and has fewer components and connectors than any other full-scale, general-purpose digital computer available for use in EW and avionics systems.

Reliability features of ATAC include:

- Maximum use of low-power Schottky TTL integrated circuits to minimize power dissipation
- Replacement of computer hardware with software, thus minimizing component count of systems in which ATAC is incorporated
- Use of standard power supply voltages to eliminate system interface reactions
- Use of minimum number of component parts, consistent with maximum performance requirements through computerized logic optimization techniques
- Minimum use of discrete components

- Packaging of all integrated circuits in standard, dual in-line ceramic-to-metal seal, hermetic packages; no circuits repackaged as hybrid devices
- Mounting of all electronic circuitry on standardized plug-in printed wiring assemblies with keyed connectors to prevent erroneous installation

5.2

AVIONICS MECHANICAL CONFIGURATION

The modular packaging design concept of the ATAC is directed toward providing the versatility of optional performance requirements and the structural and environmental integrity necessary to comply with those requirements of MIL-E-5400, class II.

The packaging design of ATAC reflects the modular packaging technique of "plug-in" subassemblies. Each module contains two printed wiring assemblies (PWA's) oriented with components facing inward. The PWA connectors are designed to NAFI requirements; interconnection can be accomplished with the standard wire wrap back panels suitable for semi-automatic wiring. The minimum ATAC consists of only two PWA's, one CPU board (ATAC-16M) and one memory board which contains control over 65K words of memory plus on-board memory (8K RAM/PROM).

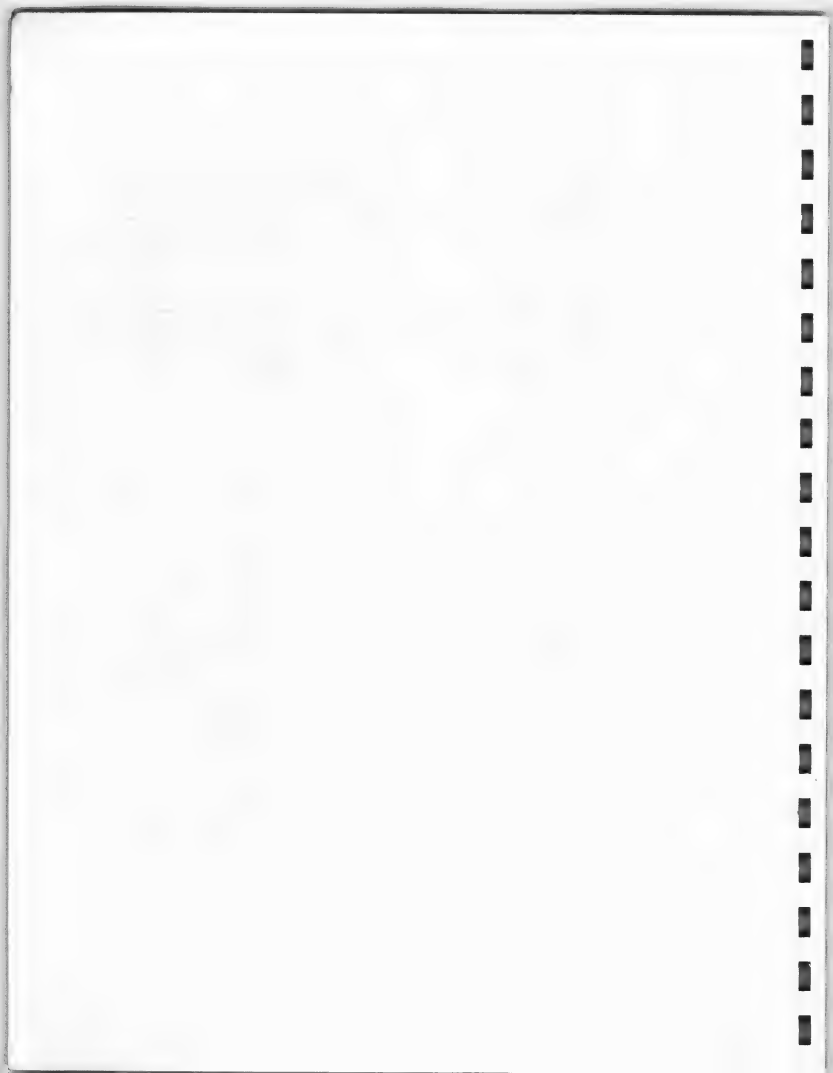
Additional single boards can be added for additional 4K or 8K of MOSRAM, up to 16K of PROM, and I/O interfaces. Additional boards for A/D or D/A conversions are readily packaged into the ATAC modular configuration.

The basic unit measures 1.8 inches wide by 6.6 inches high by 8.2 inches long, not including the wire-wrap panel, and weighs approximately 3 pounds. The estimated power dissipated in the basic unit as heat is 40 watts. Intermodular channeling is provided to allow forced air cooling where necessary. Modules of the unit can be added or deleted according to desired performance and memory required, which provides for optional arrangements (e.g., ATAC with core memory).

5.3

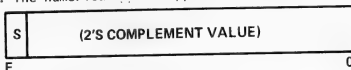
SPACE-QUALIFIED CONFIGURATION

A space-qualified version, called ATAC-16MS, has been developed with the same basic architectural features and instruction repertoire, but with a new mechanical configuration. ATAC-16MS design allows for operation in the radiation and vacuum conditions of deep space. The new mechanical design revolves around a copper thermal overlay placed on the surface of the processor. This conduction cooling system transfers the heat from the ICs to the body of the spacecraft by mechanical connection. Further information on the differences is found in the ATAC-16MS Processor Specification, document no. 20-059402.



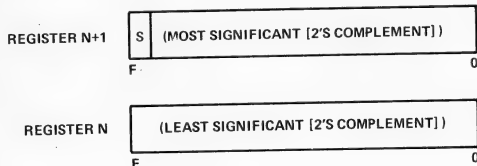
6.0 INSTRUCTION SET6.1 NUMERICAL REPRESENTATION6.1.1 Single Precision Fixed Point

A single precision numerical quantity is represented in ATAC in integer 2's complement form. The value is held in ATAC's basic 16-bit word such that bit '0' is the least significant bit. For sign determination, bit 'F' is used. The numerical value appears as follows:

6.1.2 Double Precision Fixed Point

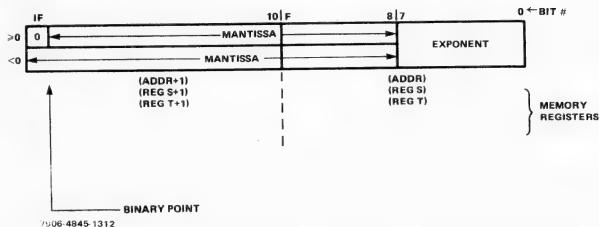
ATAC has the means to do double precision arithmetic operations such as double precision addition and subtraction. Furthermore, the multiplication and division operations involve double precision quantities in the form of product and dividend.

The representation of data as double precision in ATAC is similar to the single precision case, in that quantities are in integer 2's complement notation, and the most significant bit indicates the sign. The double precision operations require data to be loaded into two consecutive general registers, as shown below:



Floating Point Numbers and Arithmetic

A floating point number occupies two 16-bit words or registers. The sign and the most significant 15 or 16 bits of the mantissa occupy one word (register), while the balance of the mantissa together with the exponent occupies the next lower addressed word (register) as shown in the following figure:



The mantissa of a positive number is a 23-bit binary fraction, with the binary point placed as shown in the figure. The mantissa of a negative number consists of its sign bit, set to 1 and having the value of -1, plus a positive fraction appearing to the right of the binary point. The mantissa of a floating point zero is all zeros. With the execution of floating point zero, all floating point operands are assumed to be normalized; i.e., the sign bit and the first fraction bit are different. "Assumed" means that floating point operations do not check this condition and do not pre-normalize. Valid floating point operations produce post-normalized results, again with the exception of zero.

The exponent represents the power of 2 by which the mantissa must be multiplied to arrive at the value represented by the floating point number taken as a whole. Thus, for example, the number 1 is represented

$$1 = 0.4_8 \times 2^1$$

(The first hexadecimal digit of the mantissa is written as octal, since it has only 3 fraction bits.)

To avoid signed exponents and to simplify implementation, the exponent is converted to an unsigned quantity by adding 128_{10} (80_{16}) to its true algebraic value (excess-128 exponent). Thus, 00 represents 2^{-128} , 80_{16} represents 2^0 and FF_{16} represents 2^{127} ; the exponent of a floating point zero is also equal to zero.

Two fault conditions arise in floating point computations: exponent underflow and exponent overflow. Exponent underflow arises when the magnitude of a result would have a true exponent less than -128_{10} . In this case, the result is set to floating point zero and the CCR indicates a zero result indistinguishable from a true zero result. Exponent overflow arises when the magnitude of a result would have a true exponent greater than $+127_{10}$. In this case, the result is set to FFFFFFFF (not a valid floating point number, since it is not normalized) and the CCR's overflow indicator is turned on.

Some examples of floating point numbers and their evaluation are given in appendix F.

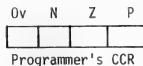
6.1.4 Floating/Fixed Conversions

Instructions have been provided which permit conversion between floating and double precision fixed point data formats utilizing programmer specified scaling. See FLT and FIX.

6.1.5 The Condition Code Register

As indicated in figure 3-4, there are three versions of the CCR: the actual hardware CCR, the 1's complement of the hardware CCR saved in the PSW (and displayed on the minipanel) and the CCR as perceived by the

programmer. All references to the CCR from this point forward will be to the latter CCR, as shown in the following figure:



Each arithmetic, logical, compare or load-type instruction which affects the CCR sets one (and only one) of the three bits N, Z or P on the basis of a 16-, 24- or 32-bit result:

- N = 1 If and only if the left-most bit of the result = 1
- Z = 1 If an only if all bits of the result = 0
- P = 1 If and only if the left-most bit of the result = 0
and at least one other bit = 1

These rules apply regardless of the occurrence of arithmetic overflow. The role of Ov, the CCR's overflow bit, is explained below:

a. Fixed Point Arithmetic

In executing a fixed point addition, subtraction or comparison (a subtraction which does not save the result) it is possible to generate a result which violates the meaning of the sign bit. Three typical examples follow (all unsubscripted numbers are hexadecimal):

- 1) Add two positive numbers and get a negative result:

$$7FFF + 0001 = 8000$$

In this case, Ov=1 to indicate the sign violation and NZP=100₂ according to the rules stated above. Thus, CCR = 1100₂. The significance of Ov=1 is that it permits branching on a positive condition,

allowing comparisons of signed numbers. For example, comparing 7FFF to FFFF will cause the result shown above. The ability to branch on positive indicates that 7FFF is indeed greater than FFFF (-1).

- 2) Subtract a positive number from a negative number and get a positive result:

Compare 8000 to 0001

$8000 - 0001 = 7FFF$ (not stored)

$CCR \leftarrow 1001_2$

Here, the $CCR = 1001_2$ permits branching on negative (less-than) indicating that 8000 (-32768_{10}) is less than 1.

- 3) Add two negative numbers and get zero:

$8000 + 8000 = 0000$

$CCR \leftarrow 1010_2$

This setting of the CCR permits branching on negative (sign of the true result) and on zero.

In fixed point division, 0v is used to indicate the aborted generation of a quotient not in the range $-32768 \leq Q \leq 32767_{10}$. The NZP bits will vary, depending on where the overflow was detected in the algorithm.

b. Floating Point Arithmetic

In floating point arithmetic, 0v = 1 indicates exponent overflow; i.e., a result has been generated which is $\geq 2^{128}$. This use of 0v also pertains to exponent overflow during a fixed point to floating point conversion operation. During a floating to fixed conversion, 0v = 1 indicates a scaling violation.

c. Other

During I/O operations, 0v is used to indicate timing-out of the interface.

INSTRUCTION FORMATS

The ATAC-16M has a full set of instructions and addressing modes. The set includes instructions for fixed and floating point arithmetic operations, bit and byte manipulation, word comparisons, subroutine linkage, branching, priority interrupt control, word-parallel input/output, and the powerful word and byte compare above limits and below limits search instructions. With the exception of the 'trap', 'halt', 'ENI' and 'DSI' instructions, all instructions are expressed in one of four formats, as shown in figure 6-1.

The instructions with formats of RR, IS, and RX are all single word instructions, while the NL format implies a double word instruction. Usually, the second word of an 'NL' is used to enter a 16-bit data value; either as an address or as a literal signed two's complement numerical value. Note that the literal value entered in an 'IS' instruction is limited to 8 bits; for 8-bit literal data quantities, values must be in the range:

$$-128 = \text{<Value>} = +127$$

while the effective address of this 8-bit field when added to PCR (for a branch conditional short) has the range:

$$(\text{PCR}) - 126 = \text{<effective address>} = (\text{PCR}) + 129$$

Throughout this section, parentheses will designate 'contents of'. For example,

(REG T)

is read: contents of register 'T'. Single quotation marks will be used to indicate the value entered in an assembly instruction field such as 'Reg S'.
Instruction:

ADD R, 'REG T', 'REG S'

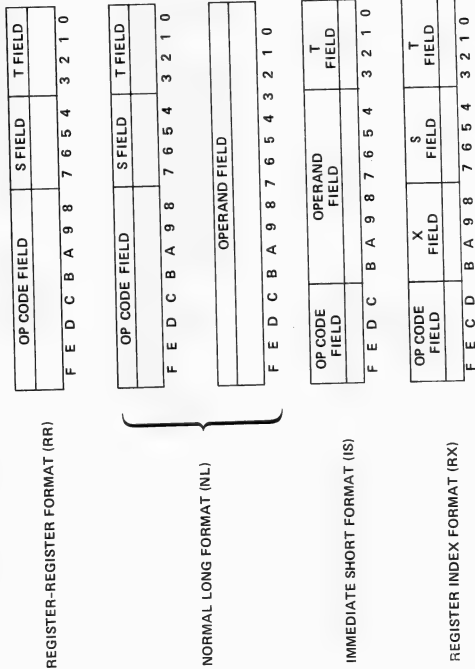


Figure 6-1. Instruction Format/Addressing Modes

In addition, a special convention is used in instructions which have a 'count' field, such as the count field in a shift instruction which counts the number of bits to be shifted. The convention is:

In all count fields, the 'count' value is one less than the number of operations to be accomplished.

****NOTE****

The actual count is entered in the assembler instruction.
The assembler decrements the actual count to generate
'count-1' in machine code.

6.3 ADDRESSING MODES

All basic instructions are modified by addressing 'modes'. The legal modes for each instruction are listed in the summary box at the beginning of the instruction description.

During ATAC-16M operation, instructions generally take source data (operands) from one of the following memory elements: the file of 16 general registers, memory, the instruction word, or an input device; and results of the operations are generally placed in one or more general registers. This rule applies to all instructions except those which specifically transfer register contents to main memory (e.g., the 'store' instructions).

ATAC provides seven basic operand addressing modes as follows:

1. Register Addressing (R Mode)

The operand is located in the register file; the register address is specified in the S or T field of the instruction word, RR format.

2. Immediate Addressing (I Mode)

The operand is located in the word immediately following the instruction word in memory; i.e., the operand word of the NL format.

3. Direct Addressing (D Mode)

The operand is located at the memory address specified in the word immediately following the instruction in memory; i.e., the operand word of the NL format.

4. Direct-Indexed Addressing (DX Mode)

The operand is located at the memory address specified by the sum of the base address (immediately following the instruction word in memory), and the contents of a general register whose address is specified in the S field of the instruction word, NL format.

5. Register-Index Addressing (RX Mode)

The operand is located at the memory address specified by the sum of the contents of two general registers whose addresses are specified in the S and X fields of the instruction word, RX format.

6. Immediate Short Addressing (IS Mode)

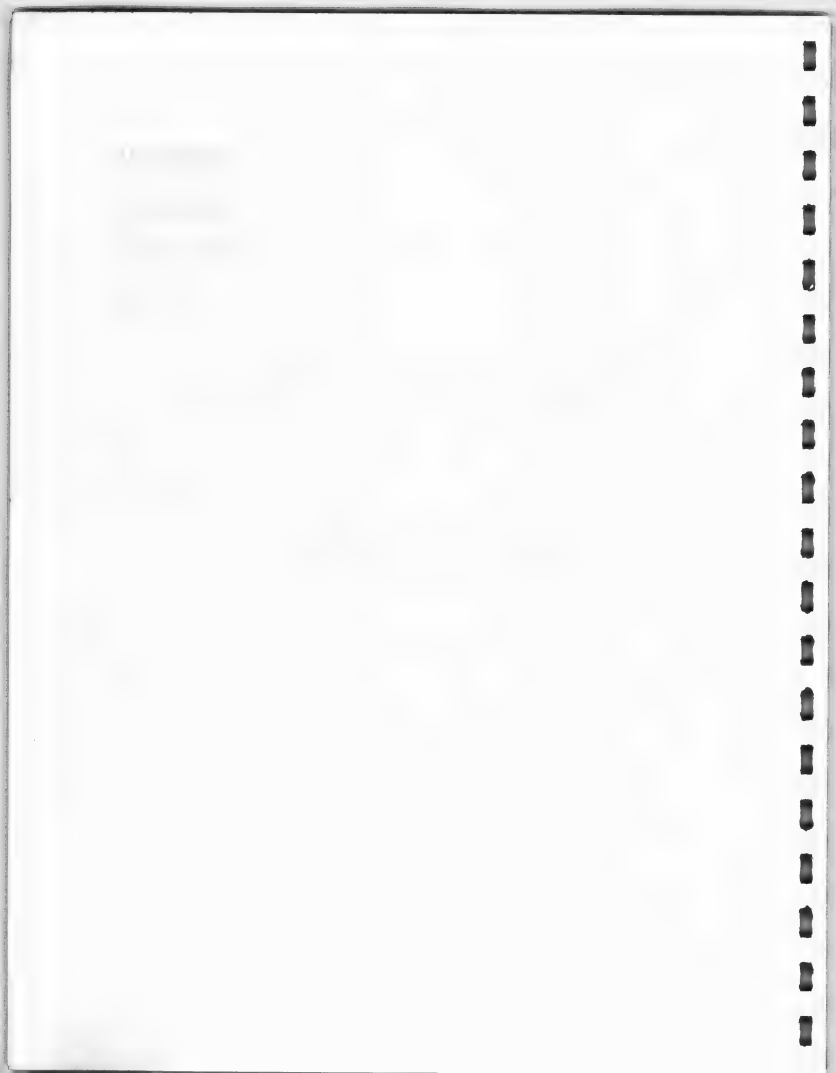
The operand is located in the operand field of the instruction word, IS format.

7. Direct Register Mode (DR Mode)

The operand is located at the memory address specified by the contents of the general register specified in the 'S' field of the instruction word, RR format.

The ATAC assembler utilizes other unique mode designator mnemonics for certain special instructions; these special instructions are described individually in the subsequent sections.

SECTION 6.4
INSTRUCTIONS



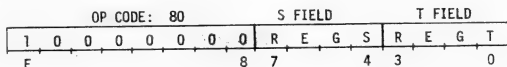
ADDITION

MODES: R, I, D, DX, IS, RX, DR

CONDITION CODE: The condition code for 'addition' is always set by the contents of the terminal register, 'REG T' as follows:

	POS	ZERO	NEG
No Overflow	0001	0010	0100
Overflow	1100	--	1001
			1010 *

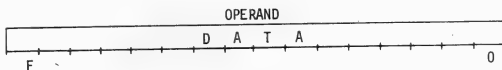
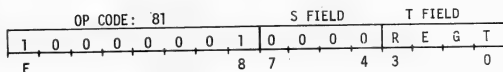
*This case occurs only when the addition is 8000 + 8000.

ADD REGISTERMODE DESIGNATOR RASSEMBLER FORMAT ADD R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The contents of 'REG T' are added to the contents of 'REG S'. The sum appears in 'REG T', and the condition code register is set.

$$(\text{REG T}) + (\text{REG T}) + (\text{REG S})$$

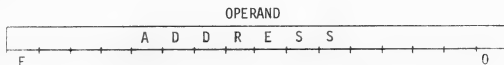
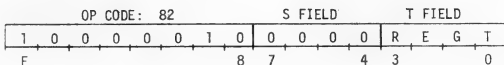
SET CCR

ADD IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT ADD I, 'REG T', 'DATA'NL FORMATDESCRIPTION

The signed, 16-bit value entered as 'data' is added to the contents of 'REG T' and the sum appears in 'REG T'. The condition code is set. The 'S' field is not used.

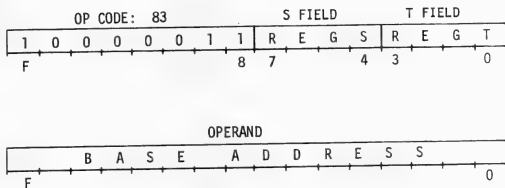
$$(\text{REG T}) \leftarrow (\text{REG T}) + \text{'DATA'}$$

SET CCR

ADD DIRECTMODE DESIGNATOR DASSEMBLER FORMAT ADD D, 'REG T', 'ADR'NL FORMATDESCRIPTION

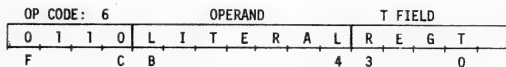
The contents of the operand data field are treated as an address. The contents of that address are added to the contents of 'REG T'. The sum appears in 'REG T', and the condition code is set. The 'S' field is not used.

(REG T) + (REG T) + (ADDRESS)
SET CCR

ADD DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT ADD DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

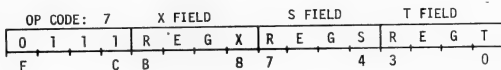
The 'base' value is added to the contents of 'REG S' to form an address. The contents of that address are added to the contents of 'REG T' and the sum appears in 'REG T'. The condition code is set.

$(REG\ T) \leftarrow (REG\ T) + ((REG\ S) + BASE)$
 SET CCR

ADD IMMEDIATE SHORTMODE DESIGNATOR ISASSEMBLER FORMAT ADD IS, 'REG T', 'LIT'IS FORMATDESCRIPTION

The signed 2's complement eight bit
 literal data value (range -128 to +127)
 is sign extended and added to 'REG T'.
 The sum appears in 'REG T', and the
 condition code is set.

(REG T) + (REG T) + 'LITERAL'
 SET CCR

ADD REGISTER INDEXEDMODE DESIGNATOR RXASSEMBLER FORMAT ADD RX, 'REG T', 'REG S', 'REG X'RX FORMATDESCRIPTION

The contents of the register specified as 'REG X' are added to the contents of 'REG S' to form an address. Both 'REG X' and 'REG S' can be used as an index or base register. The contents of the resulting address are then added to the contents of 'REG T', and the sum appears in 'REG T'. The condition code is set. The sum appears in 'REG T'. The condition code is set.

$$(\text{REG T}) \leftarrow (\text{REG T}) + ((\text{REG S}) + (\text{REG X}))$$

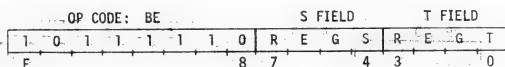
SET CCR

ADD DIRECT REGISTER

MODE DESIGNATOR DR

ASSEMBLER FORMAT ADD DR, 'REG T', 'REG S'

RR FORMAT



DESCRIPTION

The value at the address contained in 'REG S' is added to the contents of 'REG T'. The sum appears in 'REG T', and the condition code is set.

$(REG\ T) \leftarrow (REG\ T) + ((REG\ S))$
SET CCR

AND

AND

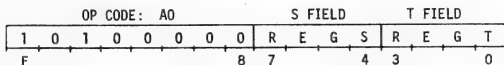
MODES: R, I, D, DX

AND Truth Table

BIT S	BIT T	RESULT
0	0	0
0	1	0
1	0	0
1	1	1

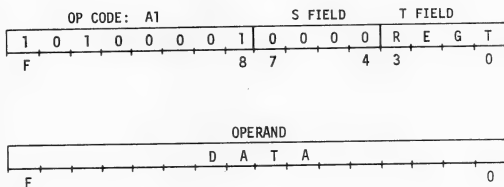
CONDITION CODE: The condition code for the 'AND' operation is always set by the contents of the terminal register, 'REG T'. The overflow is always zero.

(REG T) > 0, Condition Code is 0001
 (REG T) = 0, Condition Code is 0010
 (REG T) < 0, Condition Code is 0100

AND REGISTERMODE DESIGNATOR RASSEMBLER FORMAT AND R, 'REG T', 'REG S'RR FORMATDESCRIPTION

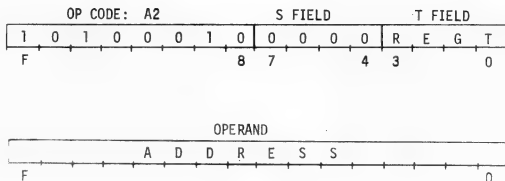
The bits in 'REG S' are 'ANDed' with the bits of 'REG T'. The logical product is placed in 'REG T', and the condition code is set.

(REG T) ← (REG T) <AND> (REG S)
SET CCR

AND IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT AND I, 'REG T', 'DATA'NL FORMATDESCRIPTION

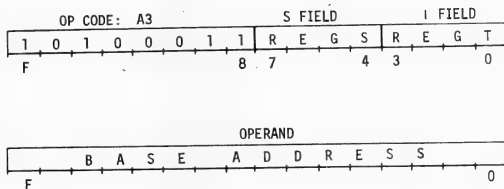
The bits of the word appearing as 'data' are 'ANDed' with the bits of 'REG T'. The logical product is placed in 'REG T', and the condition code is set. The 'S' field is not used.

(REG T) + (REG T) <AND> 'DATA'
SET CCR

AND DIRECTMODE DESIGNATOR DASSEMBLER FORMAT AND D, 'REG T', 'ADR'NL FORMATDESCRIPTION

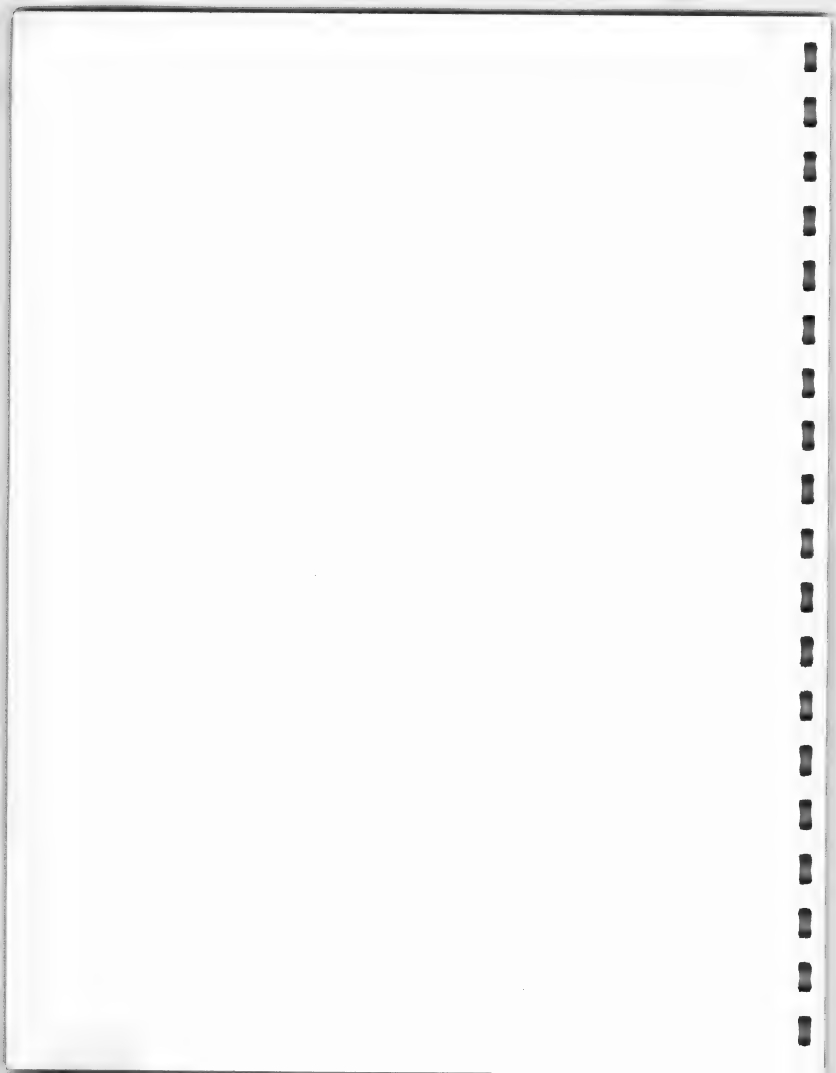
The contents of the operand data field are treated as an address. The contents of that address are 'ANDed' with the contents of 'REG T', and the logical product is placed in 'REG T'. The condition code is set. The 'S' field is not used.

(REG T) ← (REG T) <AND> (ADDRESS)
 SET CCR

AND DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT AND DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The 'base' value is added to the contents of 'REG S' to form an address. The contents of that address are 'ANDed' with the contents of 'REG T', and the logical product is placed in 'REG T'. The condition code is set.

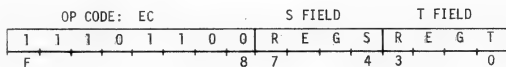
(REG T) ← (REG T) <AND> ((REG S)+BASE)
 SET CCR



BRANCH AND LINK

MODES: R, I, D, DX

CONDITION CODE: The condition code is not changed
for any of the 'BRANCH AND LINK'
type instructions.

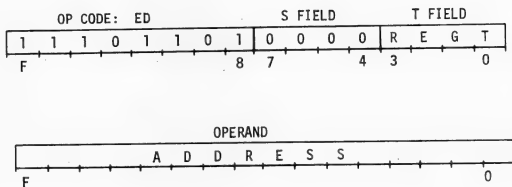
BRANCH AND LINK REGISTERMODE DESIGNATOR RASSEMBLER FORMAT BAL R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The address of the instruction following the 'branch and link' instruction is stored in 'REG T'. The program counter is then set to the contents of 'REG S'. Thus, a return can be made to the address contained in 'REG T'. The condition code is not changed.

NOTE:

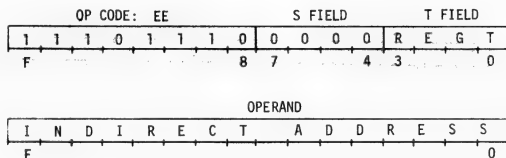
If S=T, the location of the next sequential instruction is recorded in REG S and the next sequential instruction is executed. This allows setting up a base register for position independent code.

(REG T) ← ADDRESS OF NEXT SEQUENTIAL INSTRUCTION
 (PCR) ← (REG S)

BRANCH AND LINK IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT BAL I, 'REG T', 'ADR'NL FORMATDESCRIPTION

The address of the instruction following the 'branch and link' instruction is stored in 'REG T'. The program counter is then set to the value entered as 'address', causing a branch to that address. A return can be made to the address contained in 'REG T'. The 'S' field is not used, and the condition code is not changed.

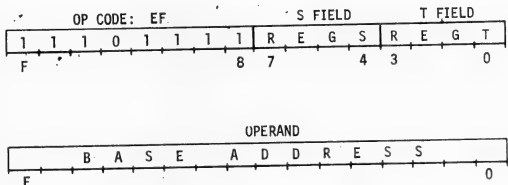
(REG T) ← ADDRESS OF NEXT SEQUENTIAL INSTRUCTION
 (PCR) ← ADDRESS

BRANCH AND LINK DIRECTMODE DESIGNATOR DASSEMBLER FORMAT BAL D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The address of the instruction following the 'branch and link' instruction is stored in 'REG T'. The value entered as operand data is treated as an address, and the contents of that address are loaded into the program counter. Thus, a branch can be made with the return address saved in 'REG T'. The condition code is not changed, and the 'S' field is not used.

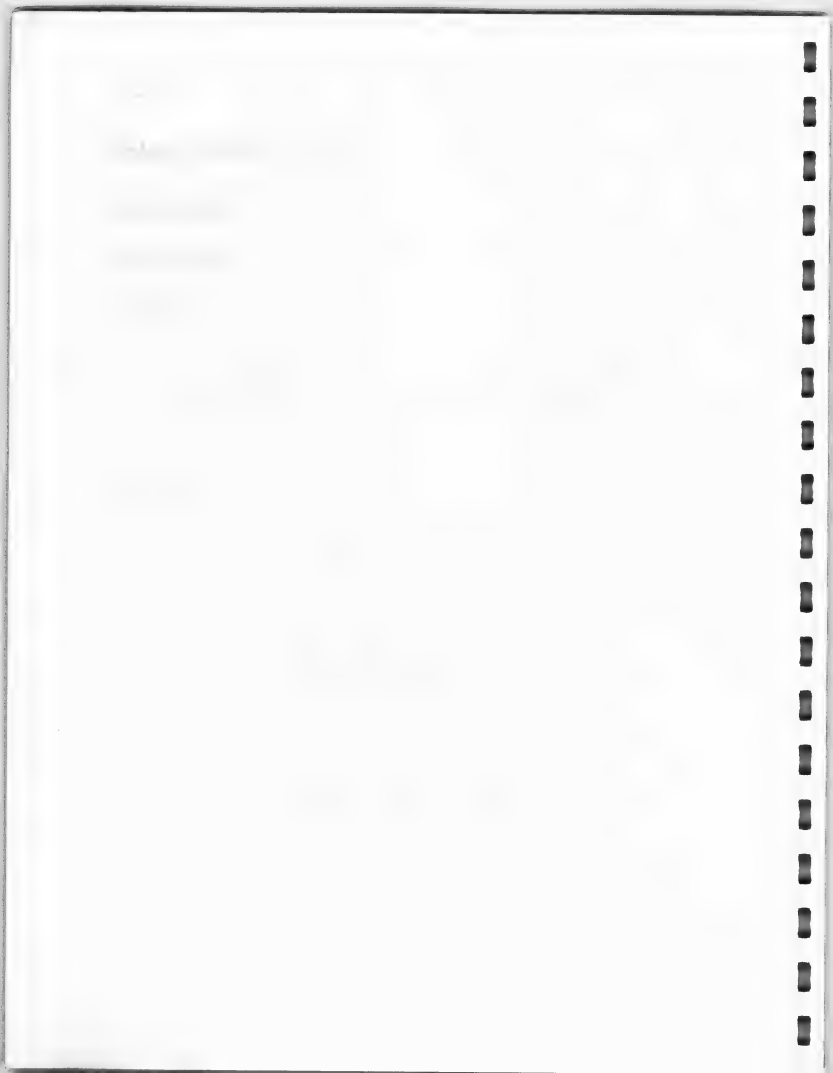
(REG T) ← ADDRESS OF NEXT SEQUENTIAL INSTRUCTION

(PCR) ← (ADDRESS)

BRANCH AND LINK DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT BAL DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The 'base' value is added to the contents of 'REG S' to form an address. The contents of that address are loaded into the program counter causing a branch. The return address is saved in 'REG T'. The condition code is not changed.

TEMP \leftarrow ((REG S) + BASE)
 (REG T) \leftarrow ADDRESS OF NEXT SEQUENTIAL INSTRUCTION
 (PCR) \leftarrow TEMP



BRANCH ON CONDITION

MODES: R, I, D, DX, IS

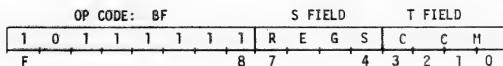
CONDITION CODE: The condition code for the branch on condition instructions is not changed.

CONDITION MATCH: The CCM/CCR match occurs as shown in the following table:

C C M BIT				CCR SETTINGS THAT WILL ALLOW BRANCHING		
3	2	1	0	Ov N Z P	Ov N Z P	Ov N Z P
X	X	X	1	0 0 0 1	1 1 0 0	
X	X	1	X	0 0 1 0	1 0 1 0	
X	1	X	X	0 1 0 0	1 0 1 0	1 0 0 1
1	X	X	X	1 1 0 0	1 0 1 0	1 0 0 1

X = DON'T CARE

PROGRAMMER NOTE: The ATAC assembler provides optional symbolic entry of branch conditions for all variations of Branch on Condition. See Appendix A.

BRANCH ON CONDITION REGISTERMODE DESIGNATOR RASSEMBLER FORMAT BRC R, 'CCM', 'REG S'RR FORMATDESCRIPTION

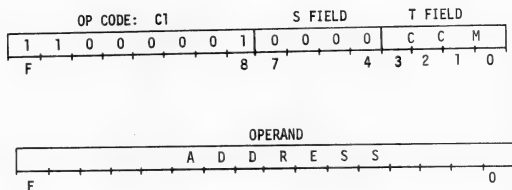
The contents of the condition code register (CCR) are compared with the four bits of the condition code mask (CCM). If any bits match*, the contents of 'REG S' are loaded into the program counter causing a branch. If no bits match*, no branch occurs, and the next instruction is executed.

TEST: (CCR) <AND> (CCM);

NO MATCH? -- EXECUTE NEXT SEQUENTIAL INSTRUCTION

ANY MATCH? -- (PCR) + (REG S)

*See table on page 6-33.

BRANCH ON CONDITION IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT BRC I, 'CCM', 'ADR'NL FORMATDESCRIPTION

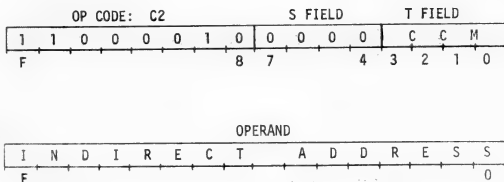
The contents of the condition code register (CCR) are compared with the four bits of the condition code mask (CCM). If any bits match*, then the value entered as 'address' is loaded into the program counter causing a branch to that location. If no bits match*, no branch occurs, and the next instruction is executed.

TEST: (CCR) <AND> (CCM);

NO MATCH? -- EXECUTE NEXT SEQUENTIAL INSTRUCTION

ANY MATCH? -- (PCR) + ADDRESS

*See table on page 6-33.

BRANCH ON CONDITION DIRECTMODE DESIGNATOR DASSEMBLER FORMAT BRC D, 'CCM', 'ADR'NL FORMATDESCRIPTION

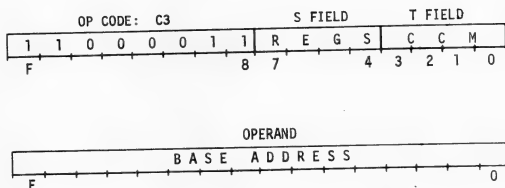
The contents of the condition code register (CCR) are compared with the four bits of the condition code mask (CCM). If any match*, then the value entered as operand is treated as an address, and the program counter is set to the contents of that address causing a branch. If no bits match*, no branch occurs, and the next instruction is executed. The 'S' field is not used.

TEST: (CCR)<AND>(CCM);

NO MATCH? -- EXECUTE NEXT SEQUENTIAL INSTRUCTION

ANY MATCH? -- (PCR) ← (ADDRESS)

*See table on page 6-33.

BRANCH ON CONDITION DIRECT-INDEXMODE DESIGNATOR DXASSEMBLER FORMAT BRC DX, 'CCM', 'BASE', 'REG S'NL FORMATDESCRIPTION

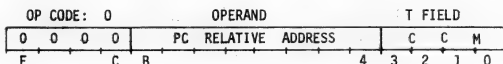
The contents of the condition code register (CCR) are compared with the four bits of the condition code mask (CCM). If any bits match*, then the value entered as 'BASE' is added to the contents of 'REG S' to form an address, and the contents of this address are loaded into the program counter. If no bits match*, no branch occurs, and next instruction is executed.

TEST: (CCR) AND (CCM);

NO MATCH? -- EXECUTE NEXT SEQUENTIAL INSTRUCTION

ANY MATCH? -- (PCR) + ((REG S)+BASE)

*See table on page 6-33.

BRANCH ON CONDITION SHORTMODE DESIGNATOR ISASSEMBLER FORMAT BRC IS, 'CCM', 'ABSOLUTE BRANCH ADDRESS'IS FORMATDESCRIPTION

The contents of the condition code register (CCR) are compared with the four bits of the condition code mask (CCM). If any bits match*, the relative address given is added to the program counter causing a branch. That relative address must use the current program counter +2 as its origin. The range of the branch is +129 to -126 from the current program counter value. If no bits match*, no branch occurs, and the next instruction is executed.

TEST: (CCR)<AND>(CCM);

NO MATCH? -- EXECUTE NEXT SEQUENTIAL INSTRUCTION

ANY MATCH? -- (PCR) + (PCR) + 'PC RELATIVE ADDRESS' + 2

*See table on page 6-33.

COMPARE BETWEEN LIMITS

MODES: R, D, DX

CBL

CONDITION CODE: The condition code for all compare between limits instructions is set the same way upon completion of instruction.

Value Below Limits: Condition Code is 0100 or 1001*

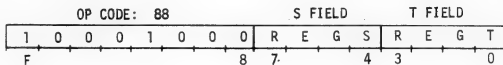
Value Within Limits: Condition Code is 0010

Value Above Limits: Condition Code is 0001 or 1110*

Range of operand and limit value is:

$-32768 \leq \text{Value} \leq 32767$

*Note: The result of the Compare Between Limits operation may set the overflow bit. In this case, the overflow indicator does not imply an error condition.

COMPARE BETWEEN LIMITS REGISTERMODE DESIGNATOR RASSEMBLER FORMAT CBL R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The content of 'REG T' is compared with the lower limit (contents of 'REG S') and the upper limit (contents of 'REG S+1'). The condition code is set. No values are modified.

(REG T) : VALUE BEING COMPARED

(REG S) : LOWER LIMIT FOR COMPARISON

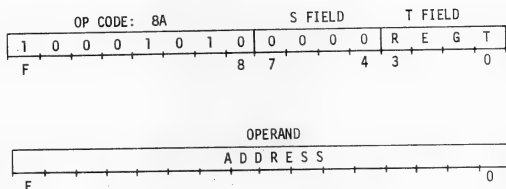
(REG S+1): UPPER LIMIT FOR COMPARISON

VALUE BELOW LIMITS: (REG T) < (REG S) ;

VALUE ABOVE LIMITS: (REG T) > (REG S +1);

VALUE WITHIN LIMITS: (REG S) ≤ (REG T) ≤ (REG S+1)

SET CCR

COMPARE BETWEEN LIMITS DIRECTMODE DESIGNATOR DASSEMBLER FORMAT CBL D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The contents of 'REG T' are compared with the lower limit value at 'ADDRESS' and the upper limit value at 'ADDRESS' +1. The condition code is set and the 'S' field is not used. No values are modified.

(REG T): VALUE BEING COMPARED

(ADR) : LOWER LIMIT OF COMPARISON

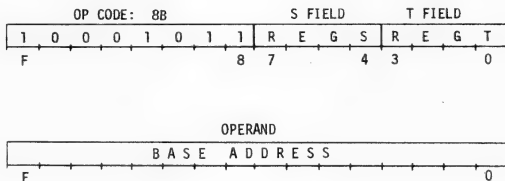
(ADR+1): UPPER LIMIT OF COMPARISON

VALUE BELOW LIMITS: (REG T) < (ADDRESS);

VALUE ABOVE LIMITS: (REG T) > (ADDRESS +1);

VALUE WITHIN LIMITS: (ADDRESS) ≤ (REG T) ≤ (ADDRESS +1)

SET CCR

COMPARE BETWEEN LIMITS-DIRECT INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT CBL DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The contents of 'REG T' are compared with the lower limit value at 'BASE' + (REG S) and the upper limit value at 'BASE' + (REG S) + 1. No values are modified, and the condition code is set.

(REG T) : VALUE COMPARED

'BASE' : BASE ADDRESS

(REG S) : INDEX TO BE USED WITH 'BASE'

(BASE+(REGS)): LOWER LIMIT FOR COMPARISON

(BASE + (REG S) + 1): UPPER LIMIT

VALUE BELOW LIMITS: (REG T) < (BASE + (REG S));

VALUE ABOVE LIMITS: (REG T) > (BASE + (REG S) + 1);

VALUE WITHIN LIMITS: (BASE + (REG S)) ≤ (REG T) ≤ (BASE + (REG S) + 1)

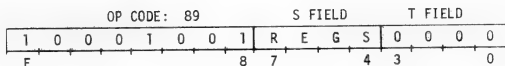
SET CCR

CINT

CLEAR INTERRUPTS

MODES: R

CONDITION CODE: The condition code is not changed
by execution of this instruction.

CLEAR INTERRUPTS-REGISTERMODE DESIGNATOR RASSEMBLER FORMAT CINT R, 'REG S'RR FORMATDESCRIPTION

This instruction clears the pending interrupts which correspond to the bits set to a '1' in the 'MASK' contained in 'REG S'. If no interrupt indicated by the 'MASK' is active, then this instruction has no effect. Bit positions 0 to 15 in the 'MASK' correspond to interrupts levels 0 to 15 (only levels 0 to 7 are provided on the one-card ATAC-16M). The interrupt mask register is not changed; the 'T' field is not used. The intent of this instruction is to clear pending interrupts regardless of the current interrupt processing level. Refer to interrupt processing, section 3.2.

COMPARE MASKED EQUALITY

MODES: R, I, D, DX

CONDITION CODE: The 'COMPARE MASKED EQUALITY' instructions set the condition code to show equality between masked fields of 'REG T' and 'REG S' as follows:

CME

EQUAL

CCR SET 0010 (ZERO)

NOT EQUAL

CCR SET 0001 (POS) When mask sign bit is '0',
or when sign bit of 'REG T'
and 'REG S' are identical.

CCR SET 0100 (NEG) When sign bits in 'REG T'
and 'REG S' differ, and
the sign bit of the mask
is '1'.

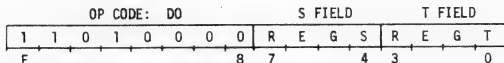
EXAMPLE OF MASKED EQUALITY

MASK	0 1 1 0 0 0 1 0 1 1 1 1 0 1 1 1
OPERAND 1	X 1 0 X X X 0 X 1 1 0 0 X 0 1 0
OPERAND 2	X 1 0 X X X 0 X 1 1 0 0 X 0 1 0
CCR Result	0 1 0

EXAMPLE OF MASKED INEQUALITY

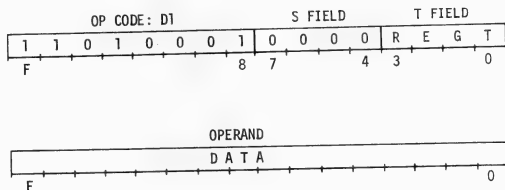
MASK	0 1 1 0 0 0 1 0 1 1 1 1 0 1 1 1
OPERAND 1	X 0 0 X X X 0 X 1 0 1 1 X 0 0 0
OPERAND 2	X 0 1 X X X 0 X 1 0 1 0 X 0 0 0
CCR Result	0 0 1

(NOTE: X may be either a '1' or '0')

COMPARE MASKED EQUALITY-REGISTERMODE DESIGNATOR RASSEMBLER FORMAT CME R, 'REG T', 'REG S'RR FORMATDESCRIPTION

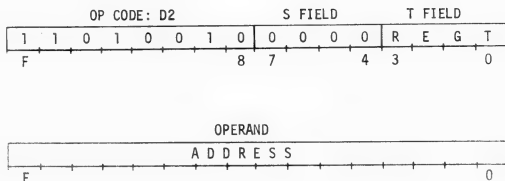
The value in 'REG T' is exclusive OR'ed with the value in 'REG S'. The results of the operation, in which a '1' occurs only when corresponding bits are not equal, are 'ANDed' with the mask in 'REG T + 1'. The condition code is set.

$((\text{REG T}) \text{ <XOR> } (\text{REG S})) \text{ <AND> } (\text{REG T}+1)$
 SET CCR

COMPARE MASKED EQUALITY -IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT CME I, 'REG T', 'DATA'NL FORMATDESCRIPTION

The value entered as 'DATA' is exclusive OR'ed with the contents of 'REG T'. The results of that operation, in which a '1' occurs only when corresponding bits are not equal, are 'ANDed' with the mask in 'REG T+1'. The condition code is set, and the 'S' field is not used.

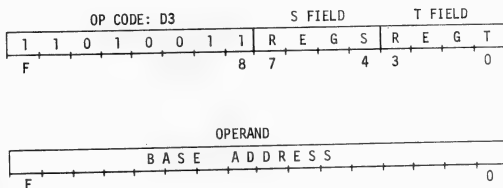
((REG T) <XOR> 'DATA') <AND> (REG T+1)
SET CCR

COMPARE MASKED EQUALITYMODE DESIGNATOR DASSEMBLER FORMAT CME D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The value found at 'ADDRESS' is exclusive OR'ed with the value in 'REG T'. The results of that operation, in which a '1' occurs only when corresponding bits are not equal, are 'ANDed' with the mask in 'REG T+1'. The condition code is not used, and the 'S' field is not used.

$$((\text{REG T}) \text{ <XOR> } (\text{ADDRESS})) \text{ <AND> } (\text{REG T+1})$$

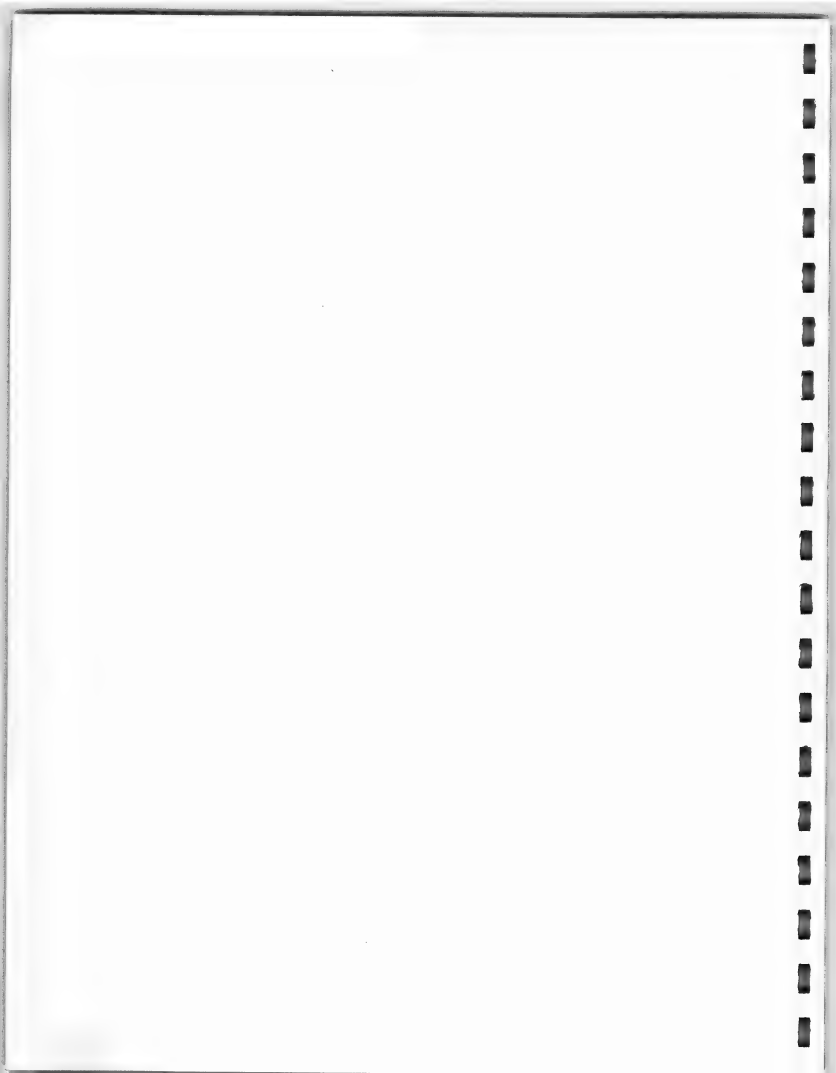
SET CCR

COMPARE MASKED EQUALITYMODE DESIGNATOR DXASSEMBLER FORMAT CME DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The value found at the address formed by adding the 'BASE ADDRESS' to the contents of 'REG S' is exclusive OR'ed with the value in 'REG T'. The results of that operation, in which a '1' occurs only when corresponding bits are not equal, are 'ANDed' with the mask in 'REG T+1'. The condition code is set.

$$((\text{REG T}) <\text{XOR}> (\text{BASE} + (\text{REG S}))) <\text{AND}> (\text{REG T}+1)$$

SET CCR

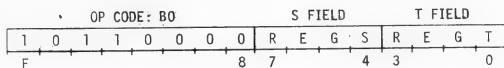


COMPARE ADDRESS**CMPA**

MODES: R, I, D, DX

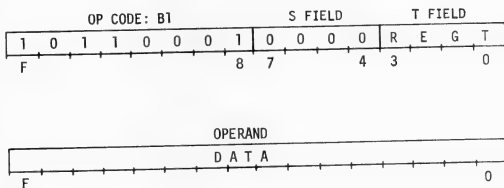
CONDITION CODE: The condition code is set based upon the results of the unsigned comparison of two 16-bit operands. This instruction provides for comparison of address values which range from 0 to $65,535_{(10)}$.

(REG T) > X [Unsigned Quantities], Condition Code is 0001
(REG T) = X [Unsigned Quantities], Condition Code is 0010
(REG T) < X [Unsigned Quantities], Condition Code is 0100

COMPARE ADDRESSMODE DESIGNATOR RASSEMBLER FORMAT CMPA R, 'REG T', 'REG S'RR. FORMATDESCRIPTION

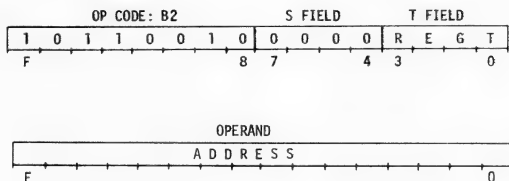
The unsigned contents of 'REG T' are compared to the unsigned contents of 'REG S' and the condition code is set. Neither 'REG S' nor 'REG T' are modified.

(REG T) VS (REG S) |Unsigned Quantities|
SET CCR

COMPARE ADDRESSMODE DESIGNATOR IASSEMBLER FORMAT CMPA I, 'REG T', 'DATA'NL FORMATDESCRIPTION

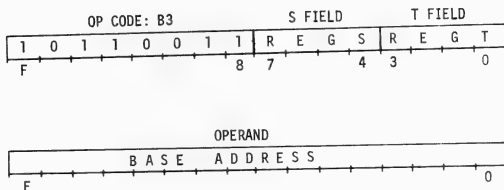
The value entered as 'DATA' is compared with the contents of 'REG T', and the condition code is set. 'REG T' contents and 'DATA' are not modified. 'S' field is not used.

(REG T) VS 'DATA' |Unsigned Quantities|
SET CCR

COMPARE ADDRESSMODE DESIGNATOR DASSEMBLER FORMAT CMPA D, 'REG T', 'ADR'NL FORMATDESCRIPTION

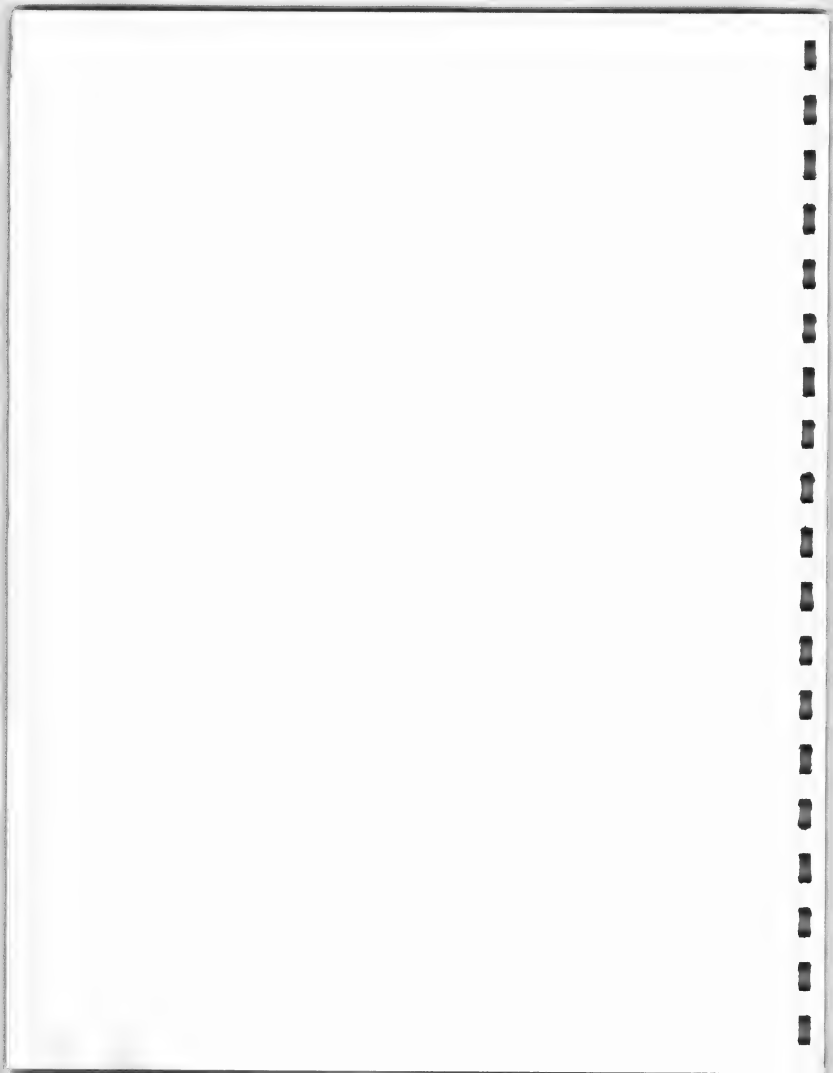
The value found at 'ADDRESS' is compared with the contents of 'REG T' and the condition code is set. The 'S' field is not used. No values are modified.

(REG T) VS (ADDRESS) | Unsigned Quantities |
SET CCR

COMPARE ADDRESSMODE DESIGNATOR DXASSEMBLER FORMAT CMPA DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The 'BASE ADDRESS' and the contents of 'REG S' are added to form an address; the contents of that address are compared with the contents of 'REG T'. The condition code is set; no values are modified.

(REG T) VS (BASE+(REG S)) |Unsigned Quantities|
SET CCR



COMPARE LOGICAL

MODES: R, I

CONDITION CODE: The 'COMPARE LOGICAL' instructions check an operand for logical equality with a mask value.

CMPL

EQUAL

CCR Set 0010

All bits set to 'one' in the mask are also 'one' in the operand.

NOT EQUAL

CCR Set 0100 (NEG) Mask sign bit is '1' and operand sign bit is '0'

CCR Set 0001 (POS) Mask sign bit is '0', or mask sign bit is '1' and operand sign bit is '1'

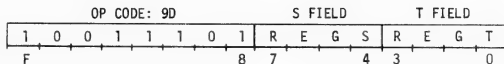
EXAMPLE OF LOGICAL EQUALITY

MASK	1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 1
OPERAND	1 1 X X X X 1 1 1 X X X 1 1 X 1
CCR RESULT	0 0 1 0

EXAMPLE OF LOGICAL INEQUALITY

MASK	1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 1
OPERAND	1 0 X X X X 1 1 0 X X X 1 1 X 1
CCR RESULT	0 0 0 1

(NOTE: X can be either '1' or '0')

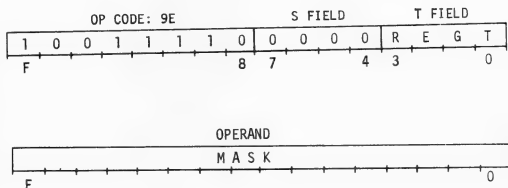
COMPARE LOGICALMODE DESIGNATOR RASSEMBLER FORMAT CMPL R, 'REG T', 'REG S'RR FORMATDESCRIPTION

This instruction tests if all bits set to 'one' in the 'REGS' mask are set to 'one' in the 'REGT' operand as follows:

The mask in 'REG S' is 'ANDed' with the one's complement of the operand in 'REG T'. If the result is zero, the masked fields are equal. The condition code is set, and no values are modified.

(REG S) <AND> (REG T)

Set CCR

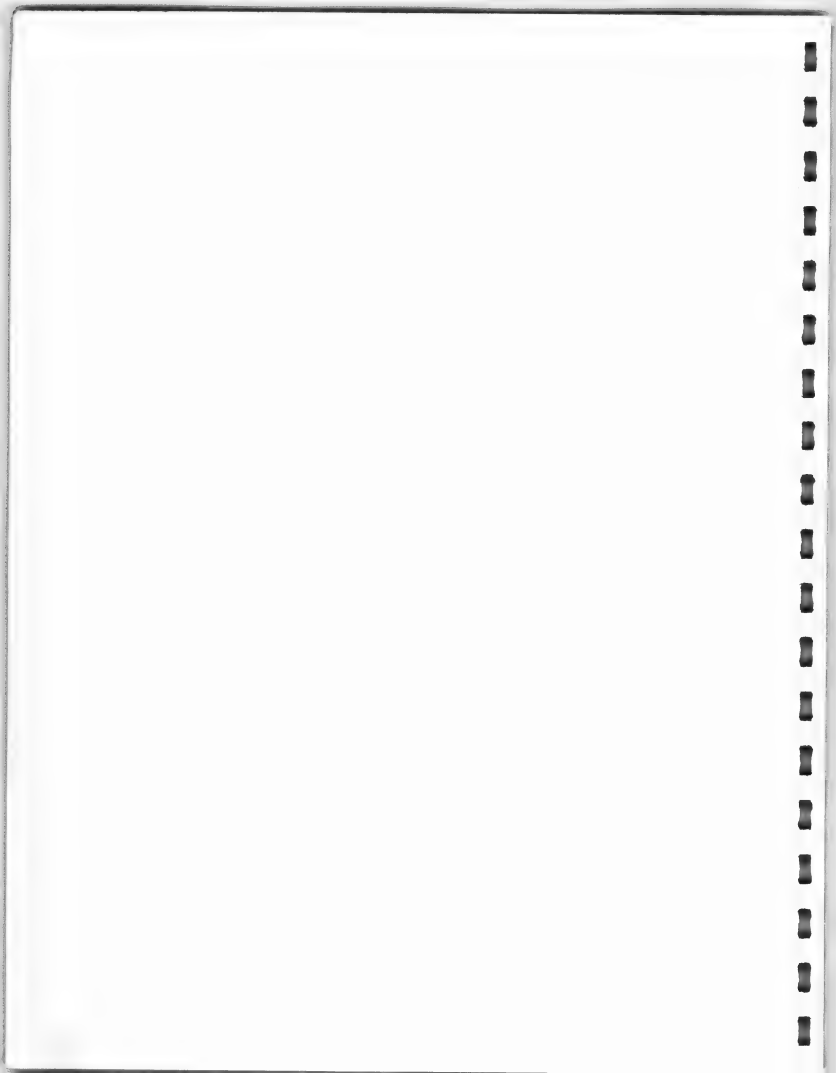
COMPARE LOGICALMODE DESIGNATOR IASSEMBLER FORMAT CMPL I, 'REG T', 'MASK'NL FORMATDESCRIPTION

This instruction tests if all bits set to 'one' in the immediate mask value are set to 'one' in the 'REG T' operand as follows:

The mask is 'ANDed' with the one's complement of the 'REG T' value. If the result is zero, the masked fields are equal. The condition code is set, no values are modified, and the 'S' field is not used.

'MASK' <AND> (REG T)

Set CCR



COMPARE SIGNED

MODES: R, I, D, DX, IS

CMPS

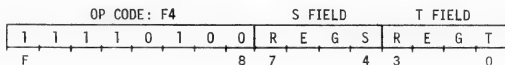
CONDITION CODE: The condition code for the compare instructions is based on the results of the comparison. If 'X' is compared with 'REG T', then the condition code is as follows:

(REG T) > X, Condition Code is 0001 or 1100*

(REG T) = X, Condition Code is 0010

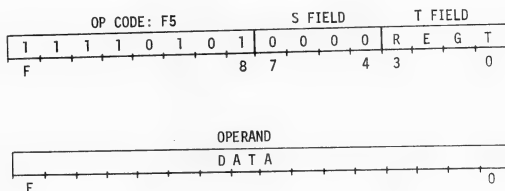
(REG T) < X, Condition Code is 0100 or 1001*

*Note: The result of the COMPARE SIGNED operation may set the overflow bit. In this case, the overflow indicator does not imply an error condition.

COMPARE SIGNEDMODE DESIGNATOR RASSEMBLER FORMAT CMPS R, 'REG T', 'REG S'RR FORMATDESCRIPTION

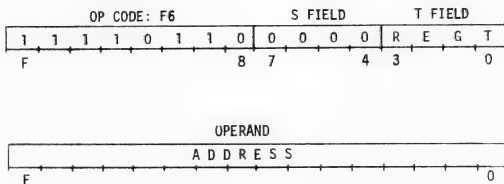
The contents of 'REG T' are compared to the contents of 'REG S' and the condition code is set. Neither 'REG S' nor 'REG T' are modified.

(REG T) VS (REG S)
SET CCR

COMPARE SIGNEDMODE DESIGNATOR IASSEMBLER FORMAT CMPS I, 'REG T', 'DATA'NL FORMATDESCRIPTION

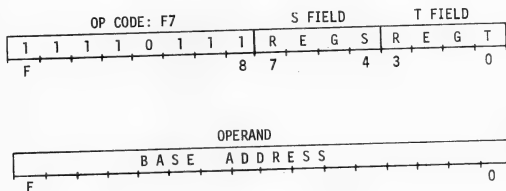
The value entered as 'DATA' is compared with the contents of 'REG T', and the condition code is set. 'REG T' contents and 'DATA' are not modified. 'S' field is not used.

(REG T) VS 'DATA'
SET CCR

COMPARE SIGNEDMODE DESIGNATOR DASSEMBLER FORMAT CMPS D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The value found at 'ADDRESS' is compared with the contents of 'REG T', and the condition code is set. The 'S' field is not used. No values are modified.

(REG T) VS (ADDRESS)
SET CCR

COMPARE SIGNEDMODE DESIGNATOR DXASSEMBLER FORMAT CMPS DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The 'BASE ADDRESS' and the contents of 'REG S' are added to form an address; the contents of that address are compared with the contents of 'REG T'. The condition code is set; no values are modified.

(REG T) VS (BASE+(REG S))
SET CCR

COMPARE SIGNEDMODE DESIGNATOR ISASSEMBLER FORMAT CMPS IS, 'REG T', 'LIT'IS FORMATDESCRIPTION

The signed value entered as literal data (RANGE -128 to +127) is right justified and sign extended to form a 16-bit value, and then compared with the contents of 'REG T'. The condition code is set, and no values are modified.

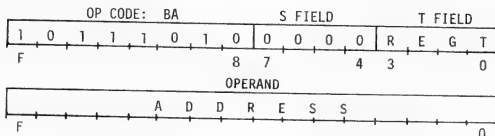
(REG T) VS 'LITERAL'
SET CCR

CLEAR MEMORY SEMAPHORE

MODES: D, DX

CONDITION CODE: The condition code is not
changed by the clear memory
semaphore instruction.

CMS

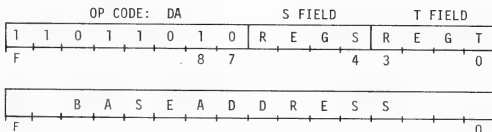
CLEAR MEMORY SEMAPHORE DIRECTMODE DESIGNATOR DASSEMBLER FORMAT CMS D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The CMS instruction provides a mechanism for releasing captured computer resources in multiprocessor configurations. Conversely, the SMS instruction is used to capture the resource for use by a processor.

The CMS locks out all other processor and I/O memory accesses for three microcycles (two containing memory references) while the multiprocessor flag bits (indicated by '1' in the corresponding bit position within REGT) of the selected semaphore word (ADR) are set to 0.

IMPLEMENTATION FLOW

<u>Step</u>	<u>Operation</u>	<u>Memory Lock</u>
1	ADR \rightarrow memory address register, read next instruction.	no
2	Read semaphore word from memory.	yes
3	Semaphore word ANDed with \rightarrow REGT; result is moved to memory output register.	yes
4	Write resulting semaphore word to memory, increment program counter.	yes
5	Fetch next instruction for execu- tion.	no

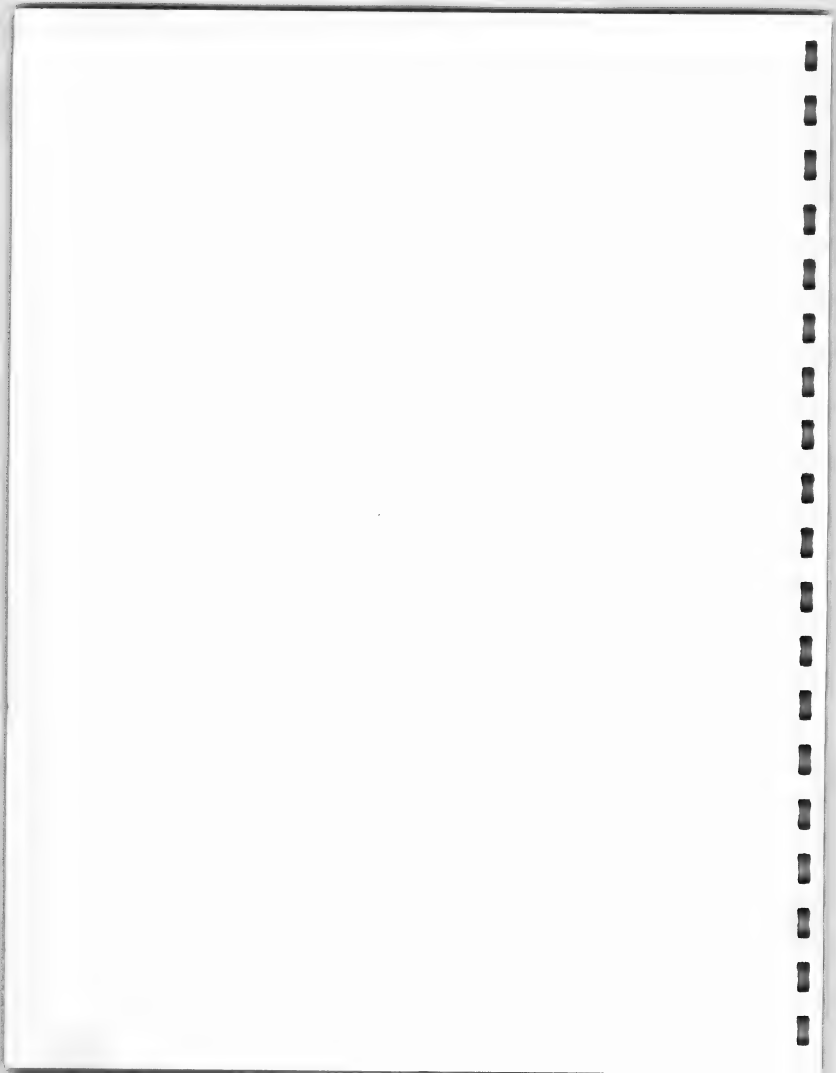
CLEAR MEMORY SEMAPHORE DIRECT INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT CMS DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The CMS instruction provides a mechanism for releasing captured computer resources in multiprocessor configurations. Conversely, the SMS instruction is used to capture the resource for use by a processor.

The CMS locks out all other processor and I/O memory accesses for three microcycles (two containing memory references) while the multiprocessor flag bits (indicated by '1' in the corresponding bit position within REGT) of the selected semaphore word (ADR) are set to 0.

IMPLEMENTATION FLOW

<u>Step</u>	<u>Operation</u>	<u>Memory Lock</u>
1	BASE + (REG S) → memory address register, read next instruction.	no
2	Read semaphore word from memory.	yes
3	Semaphore word ANDed with REGT; result is moved to memory output register.	yes
4	Write resulting semaphore word to memory, increment program counter.	yes
5	Fetch next instruction for execu- tion.	no



DOUBLE PRECISION ADDITION

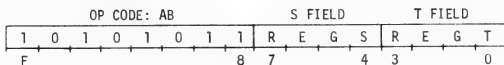
MODES: R, D

DADD

CONDITION CODE: The condition code for 'DOUBLE
PRECISION ADDITION' is set by the
32 bit sum appearing in 'REG T' + 1,
'REG T' as follows:

	POS	ZERO	NEG
No Overflow	0001	0010	0100
Overflow	1100	--	1001
			1010*

*This case occurs only when the addition is
80000000 + 80000000.

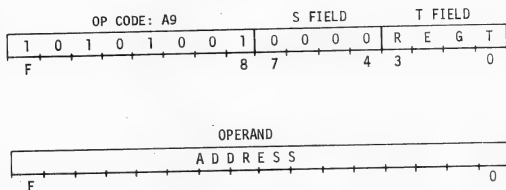
DOUBLE PRECISION ADDITION REGISTERMODE DESIGNATOR RASSEMBLER FORMAT DADD R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The contents of 'REG S+1' and 'REG S' are treated as 32 bits and added to 'REG T+1' and 'REG T'. The sum appears in 'REG T+1' and 'REG T', and the condition code is set.

$$(\text{REG T}+1) \parallel (\text{REG T}) + (\text{REG T}+1) \parallel (\text{REG T}) + \\ (\text{REG S}+1) \parallel (\text{REG S})$$

Set CCR

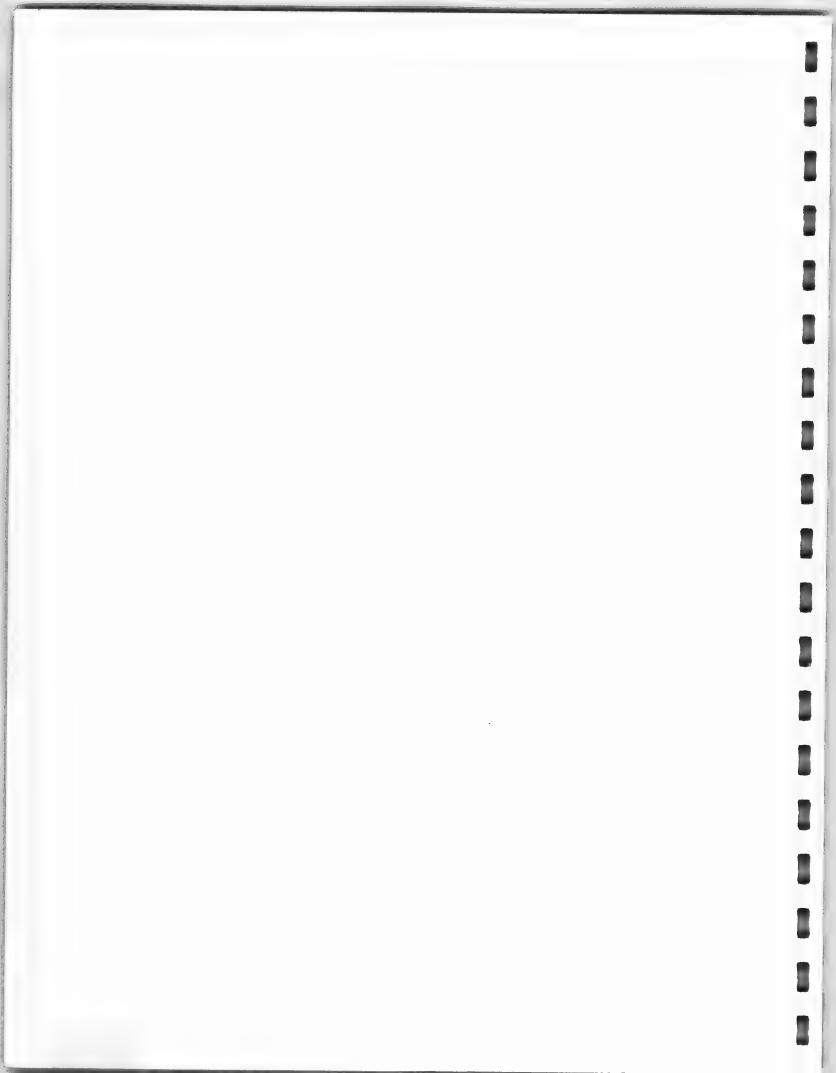
Note: T=S or T=S-1 will produce valid results; T=S+1 will not.

DOUBLE PRECISION ADDITION DIRECTMODE DESIGNATOR DASSEMBLER FORMAT DADD D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The value at 'ADDRESS+1' and 'ADDRESS' is treated as 32 bits and added to the 32 bits of 'REG T+1' and 'REG T'. The sum appears in 'REG T+1' and 'REG T', and the condition code is set. The 'S' field is not used.

$$(\text{REG T}+1) \parallel (\text{REG T}) + (\text{REG T}+1) \parallel (\text{REG T}) + \\ (\text{ADDRESS}+1) \parallel (\text{ADDRESS})$$

Set CCR



DIVISION

MODES: R

CONDITION CODE: The condition code for division depends on the quotient appearing in 'REG T+1'.

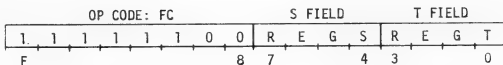
DIV

(REG T+1) > 0, Condition Code is 0001

(REG T+1) = 0, Condition Code is 0010

(REG T+1) < 0, Condition Code is 0100

(Overflow is set if the operands result in a divide error)

DIVISION REGISTERMODE DESIGNATOR RASSEMBLER FORMAT DIV R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The contents of 'REG T+1' (most significant) and 'REG T' (least significant) make up the dividend and are divided by the contents of 'REG S' (divisor). The quotient appears in 'REG T+1' and the remainder goes to 'REG T', as a positive integer. The quotient appearing in 'REG T+1' sets the condition code.

Note: S=T may produce a valid division;
S=T+1 will always cause overflow.

OVERFLOW:

Overflow is a condition produced when the dividend and divisor would yield a quotient that exceeds the magnitude allowed for a single precision quotient (i.e., $-32,768 > \text{value} > +32,767$).

If overflow occurs the results in register T, T+1 are not significant.

DISABLE INTERRUPT NETWORK

MODES: None

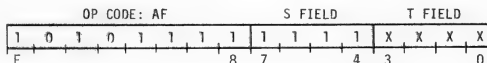
CONDITION CODE: The condition code is not
changed by the DSI instruc-
tion.

DSI

DISABLE INTERRUPT NETWORK

MODE DESIGNATOR None

ASSEMBLER FORMAT DSI

FORMATDESCRIPTION

The interrupt network is disabled and no interrupts are processed until the network is re-enabled by an ENI instruction. The interrupt mask and level are not altered. If an interrupt occurs while the network is disabled it will be captured and "remembered". This instruction does not disable the TRAP pseudo-interrupt. The contents of the T field are 'don't care'.

The contents of the S-field are an extension of the Op Code and must = 'F'. This is done automatically by the assembler. The T-field contents are "don't care" and set to 0 by the assembler.

Refer to Interrupt Processing, section 3.2.

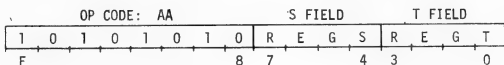
DOUBLE PRECISION SUBTRACTION

MODES: R, D

CONDITION CODE: The condition code for 'DOUBLE
PRECISION SUBTRACTION' is set by
the 32 bit difference as
follows:

	POS	ZERO	NEG
No Overflow	0001	0010	0100
Overflow	1100	--	1001

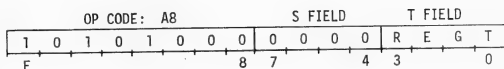
DSUB

DOUBLE PRECISION SUBTRACTION, REGISTERMODE DESIGNATOR RASSEMBLER FORMAT DSUB R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The (subtrahend) in 'REG S+1' and 'REG S' (32 bits) is subtracted from the (minuend) in 'REG T+1' and 'REG T' (32 bits). The difference appears in 'REG T+1' and 'REG T' (32 bits) and the condition code is set.

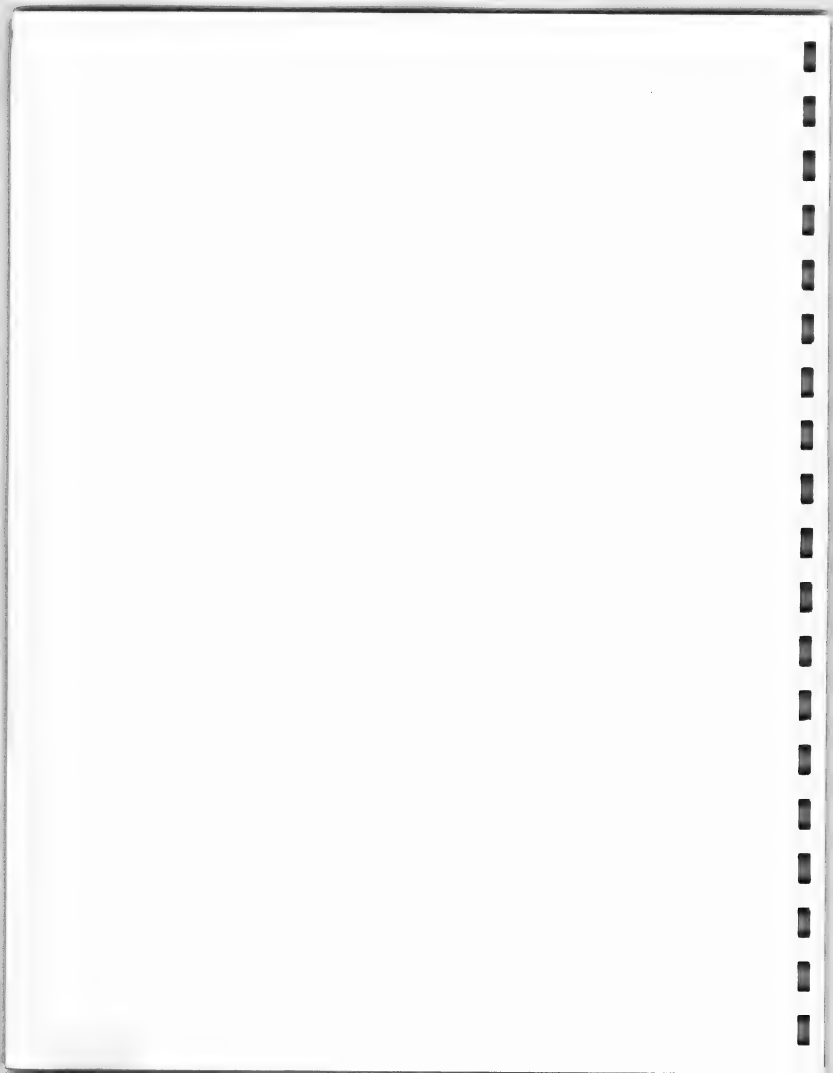
$(\text{REG T}+1) || ((\text{REG T}) + (\text{REG T}+1)) || ((\text{REG T}) - (\text{REG S}+1)) || (\text{REG S})$
 Set CCR

Note: T=S or T=S-1 will produce valid results; T=S+1 will not.

DOUBLE PRECISION SUBTRACTION, DIRECTMODE DESIGNATOR DASSEMBLER FORMAT DSUB D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The value at 'ADDRESS+1' and 'ADDRESS' (subtrahend, 32 bits) is subtracted from the 32 bits of 'REG T+1' and 'REG T' (minuend). The difference appears in 'REG T+1' and 'REG T'; the condition code is set. The 'S' field is not used.

$(\text{REG T}+1) \parallel (\text{REG T}) \leftarrow (\text{REG T}+1) \parallel (\text{REG T}) - (\text{ADDRESS}+1) \parallel (\text{ADDRESS})$
 Set CCR



ENABLE INTERRUPT NETWORK

MODES: None

CONDITION CODE: The condition code is not
changed by the ENI instruc-
tion.

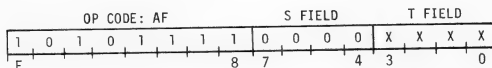
ENI

ENABLE INTERRUPT NETWORK

MODE DESIGNATOR None

ASSEMBLER FORMAT ENI

FORMAT



DESCRIPTION

The interrupt network is enabled. Any interrupt that occurs (or has occurred) is not masked and higher priority than the current level will be processed. The interrupt mask and level are not altered. The ENI instruction is not interruptable (i.e., execution of the next instruction following the ENI is assured even if an interrupt is pending).

The contents of the S-field are an extension of the Op Code and must = 0. This is done automatically by the assembler. The T-field contents are "don't care" and set to 0 by the assembler.

Refer to Interrupt Processing, Section 3.2.

FLOATING POINT ADD

MODES: R

CONDITION CODE: The condition code is set by the contents of the 24 bit mantissa result appearing in 'REG T+1', and the most significant eight bits of 'REG T' as follows:

$(\text{REG T}+1, T_{(15-8)}) > 0$, Condition Code is 0001

$(\text{REG T}+1, T_{(15-8)}) = 0$, Condition Code is 0010

$(\text{REG T}+1, T_{(15-8)}) < 0$, Condition Code is 0100

Note: On Exponent Overflow, the Condition Code is set to 1XXX and the result is set to the artificial result FFFFFFFF.

On Exponent Underflow, the Condition Code is set to 0010 and the result is set to 00000000.

FADD

FLOATING POINT DIVIDE

MODES: R

CONDITION CODE: The condition code is set by the contents of the 24 bit mantissa result appearing in 'REG T+1', and the most significant eight bits of 'REG T' as follows:

$(\text{REG T}+1, T_{(15-8)}) > 0$, Condition Code is 0001

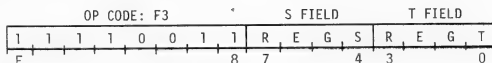
$(\text{REG T}+1, T_{(15-8)}) = 0$, Condition Code is 0010

$(\text{REG T}+1, T_{(15-8)}) < 0$, Condition Code is 0100

Note: On Exponent Overflow, the Condition Code is set to 1XXX and the result is set to the artificial result FFFFFFFF.

On Exponent Underflow, the Condition Code is set to 0010 and the result is set to 00000000.

FDIV

FLOATING POINT DIVIDEMODE DESIGNATOR RASSEMBLER FORMAT FDIV R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The floating point value in 'REG T+1' and 'REG T' is divided by the floating point value in 'REG S+1' and 'REG S'. The resulting quotient appears in 'REG T+1' and 'REG T' in floating point format. The condition code is set.

$$(\text{REG T}+1) \parallel (\text{REG T}) \leftarrow (\text{REG T}+1) \parallel (\text{REG T}) / (\text{REG S}+1) \parallel (\text{REG S})$$

floating point format

SET CCR

FIX

MODES: I

CONDITION CODE: The condition code is set by the resulting 32 bit fixed-point number appearing in 'REG T+1' and 'REG T'.

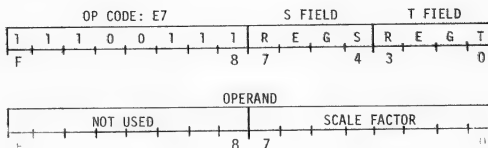
(REG T+1, REG T) > 0, Condition Code is 0001

(REG T+1, REG T) = 0, Condition Code is 0010

(REG T+1, REG T) < 0, Condition Code is 0100

Overflow, Condition Code is 1XXX

FIX

FIXMODE DESIGNATOR IASSEMBLER FORMAT FIX I, 'REG T' 'SF', 'REG S'FORMATDESCRIPTION

Convert the normalized floating point number in 'REG S+1' and 'REG S' into a fixed point number in 'REG T+1' and 'REG T'. 'REG S+1' and 'REG S' are unchanged.

The scale factor is compared with the floating operand exponent to determine the number of shifts required to scale. If the exponent of the floating point operand is larger than the scale factor indicating a left shift, the overflow indicator is set.

Note: S=T may produce a valid conversion;
S=T+1 or S=T-1 will not.

$(\text{REG T}+1) \parallel (\text{REG T}) \leftarrow (\text{REG S}+1) \parallel (\text{REG S}) \text{ Shifted right}$
 $(\text{SF} - (\text{REG S}_{(7-0)})) \text{ times}$

Set CCR

FLOAT

MODES: I

CONDITION CODE: The condition code is set by the resulting 24 bit mantissa appearing in 'REG T+1' and the most significant half of 'REG T'.

$(\text{REG T}+1, \text{REG T}_{(15-8)}) > 0$, Condition Code is 0001

$(\text{REG T}+1, \text{REG T}_{(15-8)}) = 0$, Condition Code is 0010

$(\text{REG T}+1, \text{REG T}_{(15-8)}) < 0$, Condition Code is 0100

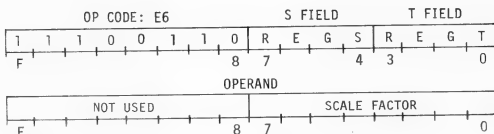
Float may result in Exponent Underflow, in which case the Condition Code = 0010 and the result = 00000000. Exponent Overflow is not possible.

FLT

FLOAT

MODE DESIGNATOR I

ASSEMBLER FORMAT FLT I, 'REG T', 'SF', 'REG S'

FORMATDESCRIPTION

Convert the 32 bit fixed-point number in 'REG S+1' and 'REG S' into a floating point number in 'REG T+1' and 'REG T'. 'REG S+1' and 'REG S' are unchanged.

The scale factor forms the base exponent before any normalization. Normalization will be performed by this instruction. To convert a fixed point 32 bit integer to a floating point number, the SF = 009F₁₆.

Note: S=T, T+1 or T-1 may all produce valid conversions.

$$(\text{REG T}+1) || (\text{REG T}_{(15-8)}) + (\text{REG S}+1) || (\text{REG S}) \text{ normalized} \\ (\text{REG T}_{(7-0)}) + \text{SF} - \# \text{ normalization steps}$$

Set CCR

FLOATING POINT MULTIPLY

MODES: R

CONDITION CODE: The condition code is set by the contents of the 24 bit mantissa result appearing in 'REG T+1', and the most significant eight bits of 'REG T' as follows:

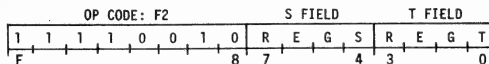
$(\text{REG T}+1, T_{(15-8)}) > 0$, Condition Code is 0001

$(\text{REG T}+1, T_{(15-8)}) = 0$, Condition Code is 0010

$(\text{REG T}+1, T_{(15-8)}) < 0$, Condition Code is 0100

Note: On Exponent Overflow, the Condition Code is set to 1XXX and the result is set to the artificial result FFFFFFFF.

On Exponent Underflow, the Condition Code is set to 0010 and the result is set to 00000000.

FLOATING POINT MULTIPLYMODE DESIGNATOR RASSEMBLER FORMAT FMUL R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The floating point value in 'REG T+1' and 'REG T' is multiplied by the floating point value in 'REG S+1' and 'REG S'. The resulting product appears in 'REG T+1' and 'REG T'. The condition code is set.

$$(\text{REG T}+1) || (\text{REG T}) + (\text{REG T}+1) || (\text{REG T}) * (\text{REG S}+1) || (\text{REG S})$$
 floating point format

Set CCR

FLOATING POINT SUBTRACT

MODES: R

CONDITION CODE: The condition code is set by the contents of the 24 bit mantissa result appearing in 'REG T+1', and the most significant eight bits of 'REG T' as follows:

$(\text{REG T}+1, T_{(15-8)}) > 0$, Condition Code is 0001

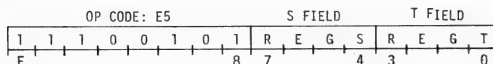
$(\text{REG T}+1, T_{(15-8)}) = 0$, Condition Code is 0010

$(\text{REG T}+1, T_{(15-8)}) < 0$, Condition Code is 0100

Note: On Exponent Overflow, the Condition Code is set to 1XXX and the result is set to the artificial result FFFFFFFF.

On Exponent Underflow, the Condition Code is set to 0010 and the result is set to 00000000.

FSUB

FLOATING POINT SUBTRACTMODE DESIGNATOR RASSEMBLER FORMAT FSUB R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The floating point value in 'REG S' and 'REG S+1' is subtracted from the floating point value in 'REG T' and 'REG T+1'.

The floating point result appears in 'REG T' and 'REG T+1'. The condition code register is set.

$$(\text{REG T}+1) \leftarrow ((\text{REG T}) \leftarrow (\text{REG T}+1)) \leftarrow ((\text{REG T}) - (\text{REG S}+1)) \leftarrow (\text{REG S})$$

floating point format

Set CCR

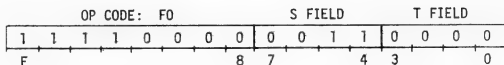
HALT

MODES: NO MODES

CONDITION CODE: Execution of the 'HALT' instruction
does not change the condition code.

Note: Halts when minipanel is attached.
Acts as a NOP when minipanel is
not attached.

HALT

HALTMODE DESIGNATOR NO MODESASSEMBLER FORMAT HALTRR FORMATDESCRIPTION

With the control panel connected, the 'HALT' instruction causes the computer to be stopped. Control functions may then be executed from the control panel.

With the control panel disconnected, the 'HALT' instruction is treated as a 'NOP'.

Note: The S and T-field values as shown are required and are provided automatically by the assembler.

INCREMENT AND BRANCH NEGATIVE

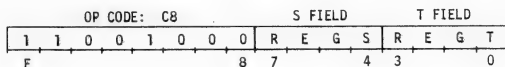
MODES: R, I, D, DX

CONDITION CODE: The condition code is set to show the status of the register which is incremented, as follows:

	POS	ZERO	NEG
No Overflow	0001	0010	0100
Overflow	1100*	--	--

*This condition occurs when 7FFF is incremented to 8000. This will not cause a branch.

IBN

INCREMENT AND BRANCH NEGATIVE REGISTERMODE DESIGNATOR RASSEMBLER FORMAT IBN R, 'REG T', 'REG S'RR FORMATDESCRIPTION

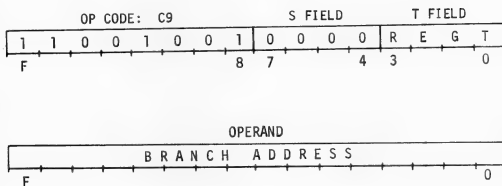
The contents of 'REG T' are incremented and stored back in 'REG T', and then tested. If the contents of 'REG T' are greater than or equal to zero, the next instruction is executed. If the contents of 'REG T' are less than zero, the program counter is set to the contents of 'REG S' causing a branch to that location.

(REG T) ← (REG T) + 1; Set CCR;

TEST: (REG T) < 0?

NO, EXECUTE NEXT SEQUENTIAL INSTRUCTION

YES, (PCR) ← (REG S)

INCREMENT AND BRANCH NEGATIVE, IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT IBN I, 'REG T', 'ADR'NL FORMATDESCRIPTION

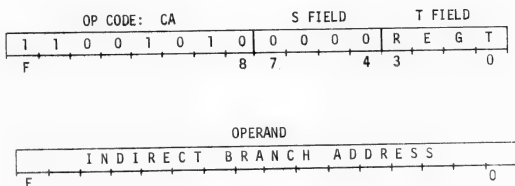
The contents of 'REG T' are incremented and stored back into 'REG T', and then tested. If the contents of 'REG T' are greater than or equal to zero, the next instruction is executed. If the contents of 'REG T' are less than zero, the program counter is set to the value 'ADDRESS' causing a branch to that location.

(REG T) ← (REG T) + 1 ; Set CCR;

TEST: (REG T) < 0?

NO, EXECUTE NEXT SEQUENTIAL INSTRUCTION

YES, (PCR) ← 'ADDRESS'

INCREMENT AND BRANCH NEGATIVE DIRECTMODE DESIGNATOR DASSEMBLER FORMAT IBN D, 'REG T', 'ADR'NL FORMATDESCRIPTION

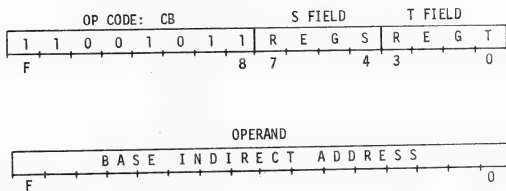
The contents of 'REG T' are incremented and stored back in 'REG T', and then tested. If the contents of 'REG T' are greater than or equal to zero, the next instruction is executed. If the contents of 'REG T' are less than zero, the value found at the address entered as 'ADR' is loaded into the program counter causing a branch.

(REG T) \leftarrow (REG T) + 1 ; Set CCR;

TEST: (REG T) < 0?

NO, EXECUTE NEXT SEQUENTIAL INSTRUCTION

YES, (PCR) \leftarrow (ADDRESS)

INCREMENT AND BRANCH NEGATIVE DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT IBN DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

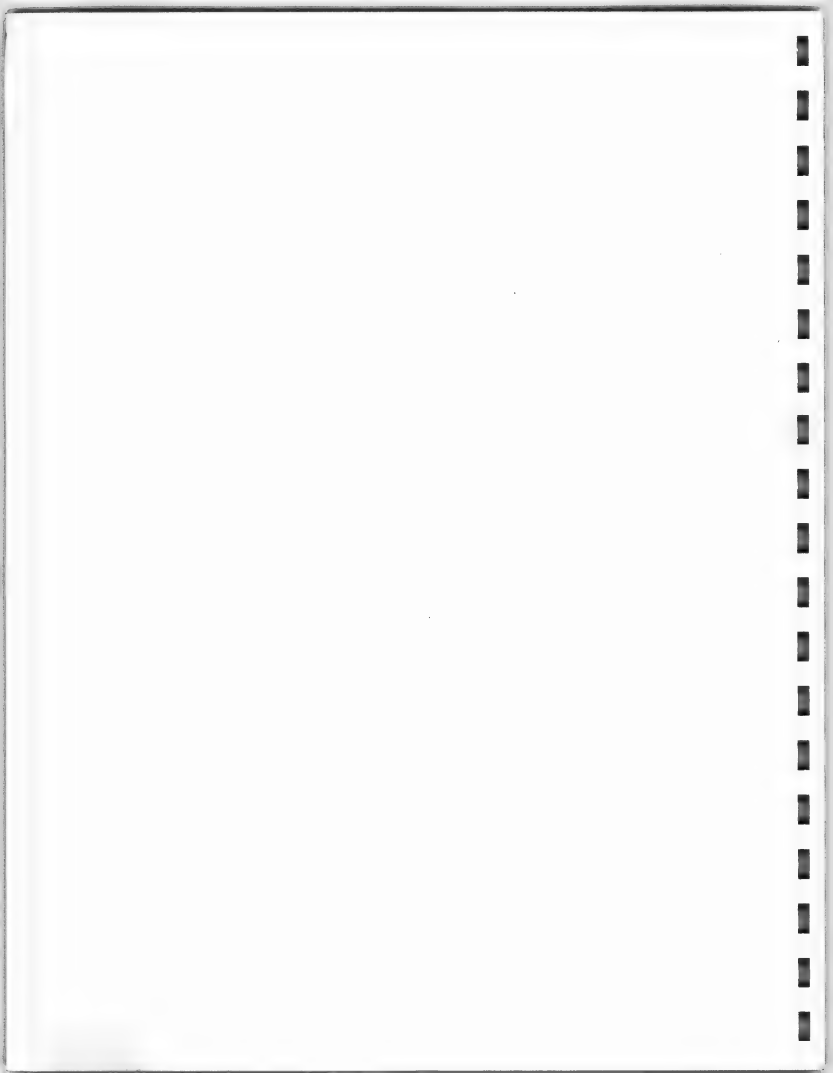
The contents of 'REG T' are incremented and stored back in 'REG T', and then tested. If the contents of 'REG T' are greater than or equal to zero, the next instruction is executed. If the value in 'REG T' is less than zero, the content of the address formed by adding 'BASE' and 'REG S' is put into the program counter causing a branch to that location.

(REG T) ← (REG T) + 1 ; Set CCR;

TEST: (REG T) < 0?

NO, EXECUTE NEXT SEQUENTIAL INSTRUCTION

YES, PCR ← (BASE+(REG S))



INCLUSIVE OR

MODES: R, I, D, DX

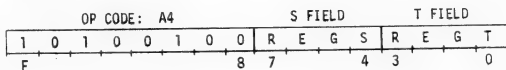
INCLUSIVE-OR TRUTH TABLE		
BIT S	BIT T	RESULT
0	0	0
0	1	1
1	0	1
1	1	1

CONDITION CODE: The condition code for inclusive-or instructions is based on the final results appearing in 'REG T' as follows. Overflow is always zero.

(REG T) > 0, Condition Code is 0001

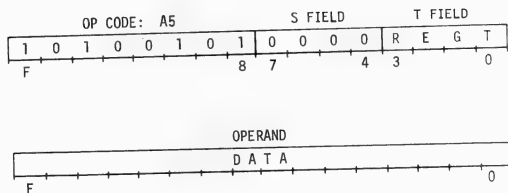
(REG T) = 0, Condition Code is 0010

(REG T) < 0, Condition Code is 0100

INCLUSIVE OR REGISTERMODE DESIGNATOR RASSEMBLER FORMAT IOR R, 'REG T', 'REG S'RR FORMATDESCRIPTION

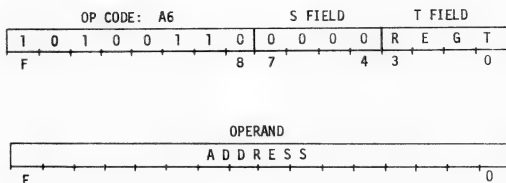
The contents of 'REG S' are inclusive OR'ed with the contents of 'REG T'. The results appear in 'REG T' and the condition code is set.

(REG T) ← (REG T) <OR> (REG S)
Set CCR

INCLUSIVE OR IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT IOR I, 'REG T', 'DATA'NL FORMATDESCRIPTION

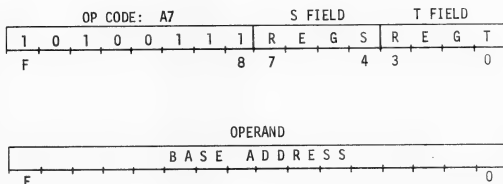
The value entered as 'DATA' is inclusive OR'ed with the content of 'REG T'. The results appear in 'REG T' and the condition code is set. The 'S' field is not used.

(REG T) ← (REG T) <OR> 'DATA'
Set CCR

INCLUSIVE OR DIRECTMODE DESIGNATOR DASSEMBLER FORMAT IOR D, 'REG T', 'ADR'NL FORMATDESCRIPTION

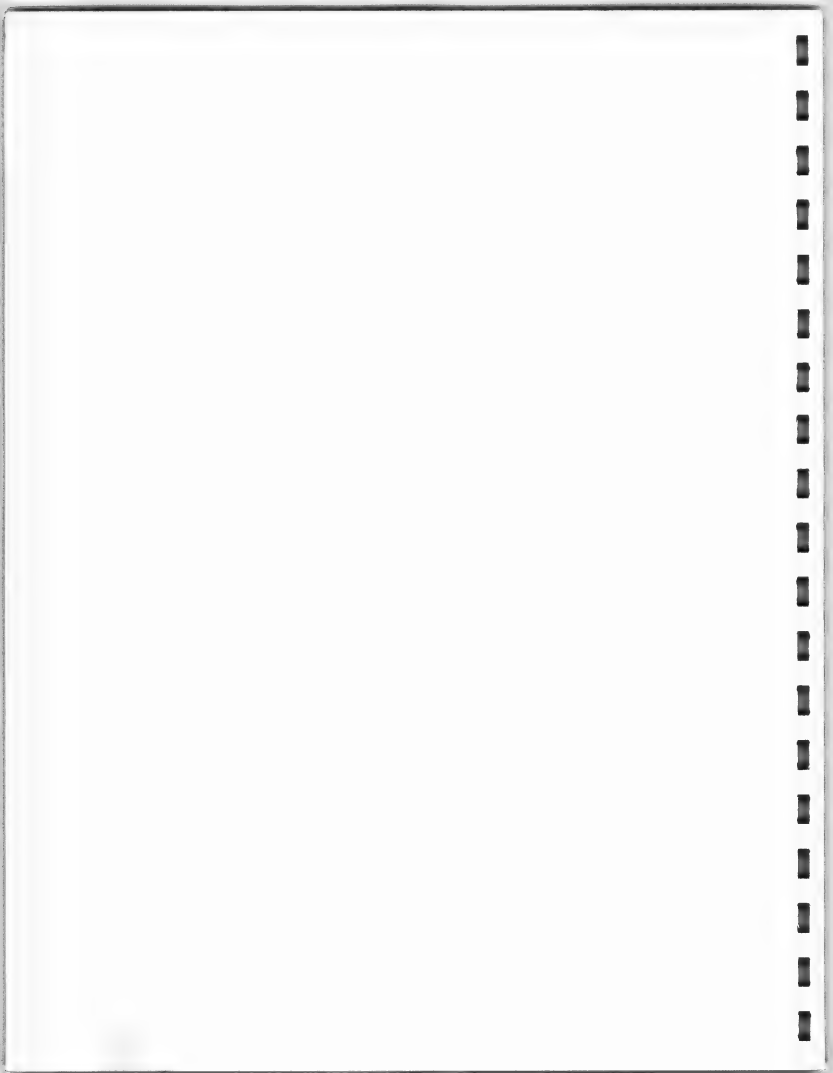
The contents of 'ADDRESS' are inclusive OR'ed with the contents of 'REG T'. The results appear in 'REG T' and the condition code is set. The 'S' field is not used.

(REG T) ← (REG T) <OR> (ADDRESS)
Set CCR

INCLUSIVE-OR DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT IOR DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The content of the address formed by adding the value in 'BASE' to the contents of 'REG S' are inclusive OR'ed with the contents of 'REG T'. The results appear in 'REG T', and the condition code is set.

$(\text{REG T}) \leftarrow (\text{REG T}) \text{ <OR> } (\text{BASE} + (\text{REG S}))$
 SET CCR



LOAD ARITHMETIC (TWO'S) COMPLEMENT

MODES: R, I, D, DX

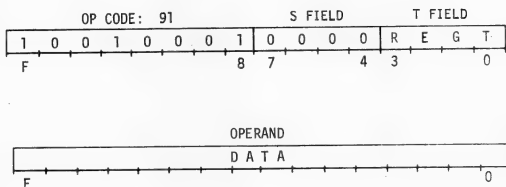
CONDITION CODE: The condition code for 'LOAD
ARITHMETIC COMPLEMENT' instructions
is based on the final contents of
'REG T' as follows:

(REG T) > 0, Condition Code is 0001

(REG T) = 0, Condition Code is 0010

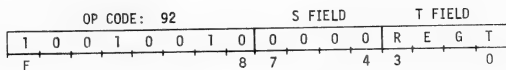
(REG T) < 0, Condition Code is 0100

OVF is set if the maximum negative 2's
complement number is loaded.

LOAD ARITHMETIC (TWO'S) COMPLEMENT IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT LAC I, 'REG T', 'DATA'NL FORMATDESCRIPTION

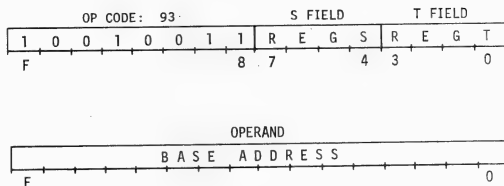
The two's complement of the value entered as 'DATA' is loaded into 'REG T', and the condition code is set. The 'S' field is not used.

(REG T) ← TWO'S COMP OF 'DATA'
 SET CCR

LOAD ARITHMETIC (TWO'S) COMPLEMENT DIRECTMODE DESIGNATOR DASSEMBLER FORMAT LAC D, 'REG T', 'ADR'NL FORMATDESCRIPTION

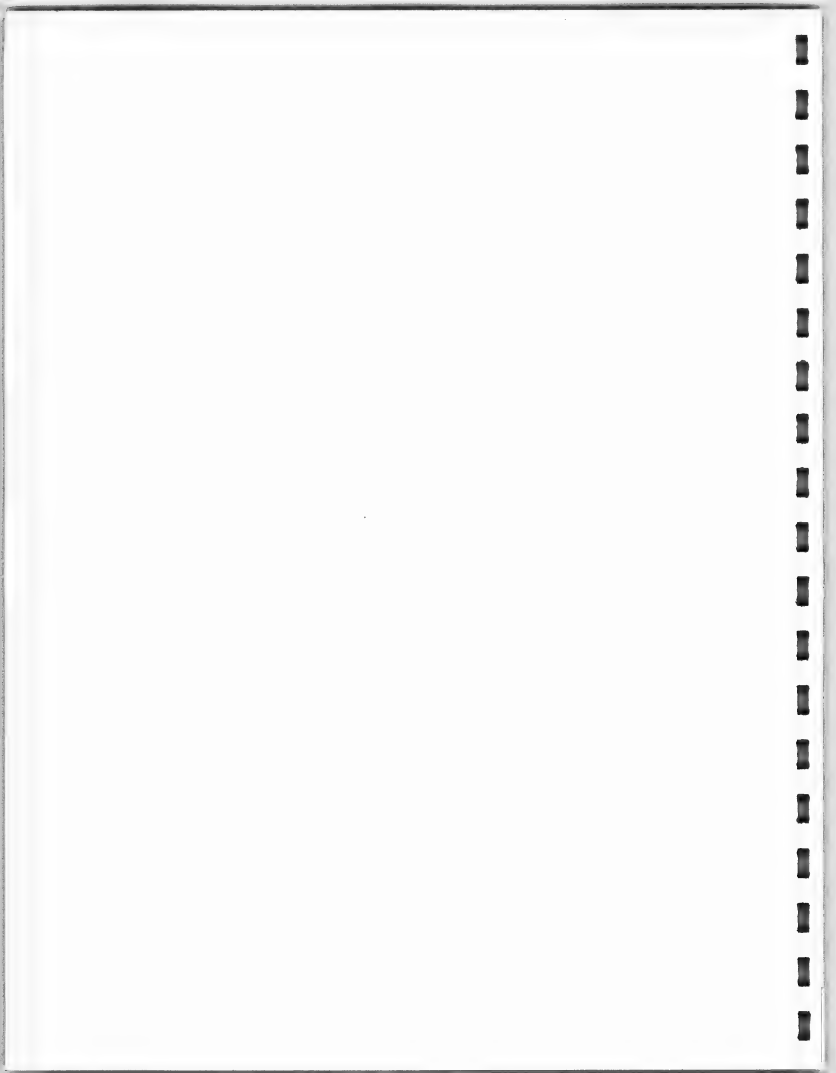
The two's complement of the contents of 'ADDRESS' is loaded into 'REG T' and the condition code is set. The 'S' field is not used.

(REG T) ← TWO'S COMP OF (ADDRESS)
SET CCR

LOAD ARITHMETIC (TWO'S) COMPLEMENT DIRECT INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT LAC DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The two's complement of the content of the address formed by adding 'BASE' to the contents of 'REG S' is loaded into 'REG T'. The condition code is set.

$(REG\ T) \leftarrow TWO'S\ COMP\ OF\ (BASE + (REG\ S))$
 SET CCR



LOAD LOWER BYTE

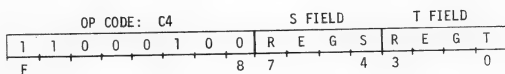
MODES: R, I, D, DX

CONDITION CODE: The condition code for 'LOAD LOWER
BYTE' is always set by the final
contents of 'REG T' as follows:

(REG T) > 0, Condition Code is 0001

(REG T) = 0, Condition Code is 0010

Condition Code 0100 is not used

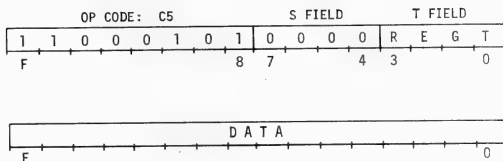
LOAD LOWER BYTE, REGISTERMODE DESIGNATOR RASSEMBLER FORMAT LDLB R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The low order byte (BITS 7-0) of the contents of 'REG S' are loaded into the low order byte of 'REG T.' The condition code is set.

$$(\text{REG T}, 7-0) \leftarrow (\text{REG S}, 7-0)$$

$$(\text{REG T}, F-8) \leftarrow 0$$

SET CCR

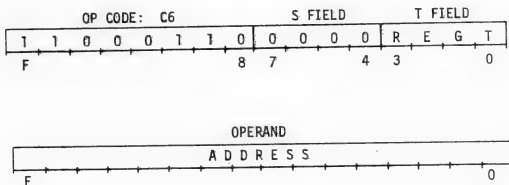
LOAD LOWER BYTE, IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT LDLB I, 'REG T', 'DATA'NL FORMATDESCRIPTION

The lower byte (bits 7-0) of the value entered as 'DATA' is loaded into the low order byte of 'REG T'. The condition code is set, and the 'S' field is not used.

(REG T, 7-0) ← 'DATA', BITS 7-0

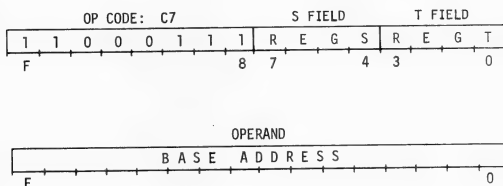
(REG T, F-8) ← 0

SET CCR

LOAD LOWER BYTE DIRECTMODE DESIGNATOR DASSEMBLER FORMAT LDLB D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The low order byte of the contents of 'ADDRESS' (bits 7-0) is loaded into the low order byte of 'REG T.' The condition code is set. The 'S' field is not used.

(REG T, 7-0) + (ADDRESS) BITS 7-0
 (REG T, F-8) + 0
 SET CCR

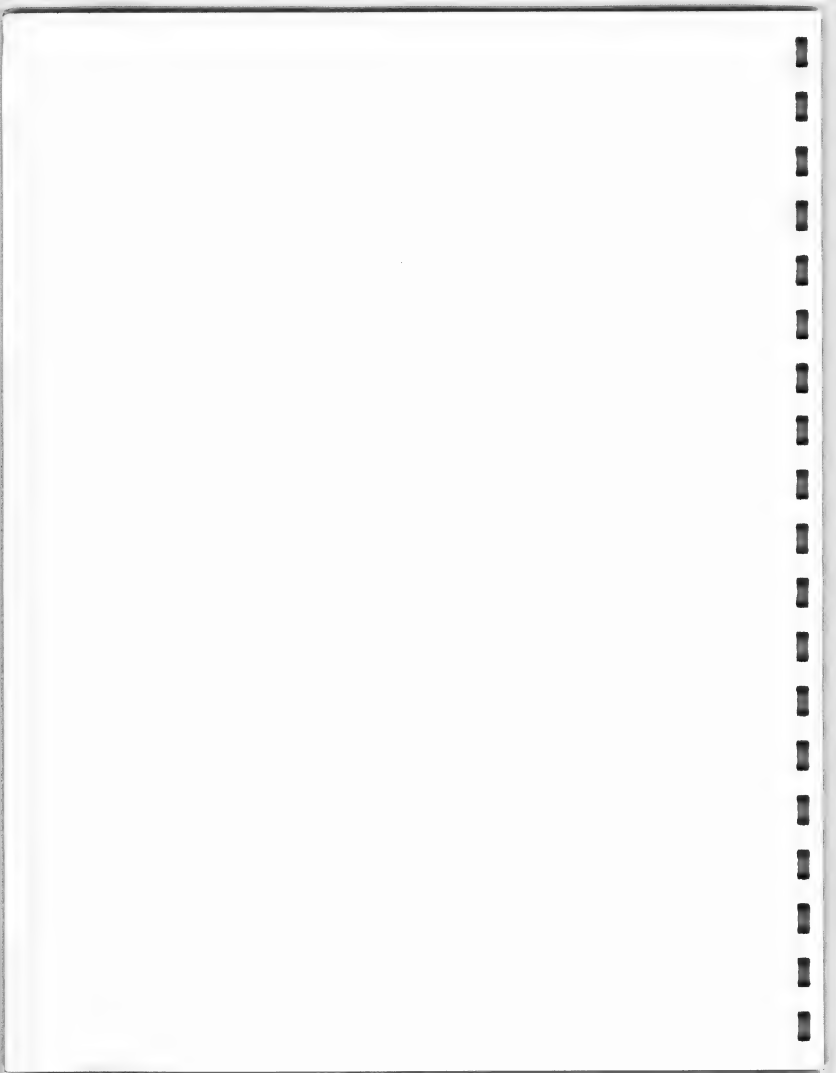
LOAD LOWER BYTE DIRECT INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT LDLB DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The value entered as 'BASE' is treated as a base address, and the contents of 'REG S' are taken as an index with respect to that base. The low order byte of the contents of the address formed by adding 'BASE' and 'REG S' is loaded into the low order byte of 'REG T.' The condition code is set.

$$(\text{REG T}, 7-0) \leftarrow (\text{BASE} + (\text{REG S})) \text{ BITS } 7-0$$

$$(\text{REG T}, \text{F}-8) \leftarrow 0$$

SET CCR



LOAD-REGISTER

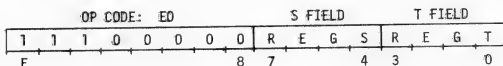
MODES: R, I, D, DX, IS, RX

CONDITION CODE: The condition code is set by the value loaded into 'REG T' as follows:

(REG T) > 0, Condition Code is 0001

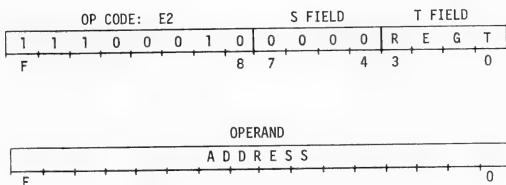
(REG T) = 0, Condition Code is 0010

(REG T) < 0, Condition Code is 0100

LOAD REGISTER, REGISTERMODE DESIGNATOR RASSEMBLER FORMAT LDR R, 'REG T', 'REG S'RR FORMATDESCRIPTION

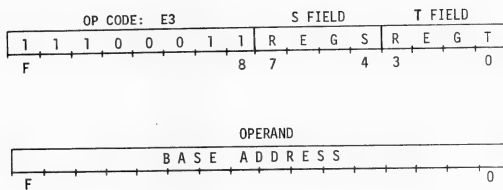
The contents of 'REG S' are loaded into
'REG T'. The condition code is set.

(REG T) ← (REG S)
SET CCR

LOAD-REGISTER, DIRECTMODE DESIGNATOR DASSEMBLER FORMAT LDR D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The contents of 'ADDRESS' are loaded into 'REG T.' The condition code is set. The 'S' field is not used.

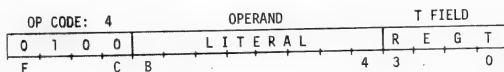
(REG T) ← (ADDRESS)
 SET CCR

LOAD-REGISTER, DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT LDR DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The contents of the address formed by adding 'BASE' and 'REG S' are loaded into 'REG T'. The condition code is set.

$$(\text{REG T}) \leftarrow (\text{BASE} + (\text{REG S}))$$

SET CCR

LOAD-REGISTER, IMMEDIATE SHORTMODE DESIGNATOR ISASSEMBLER FORMAT LDR IS, 'REG T', 'LIT'IS FORMATDESCRIPTION

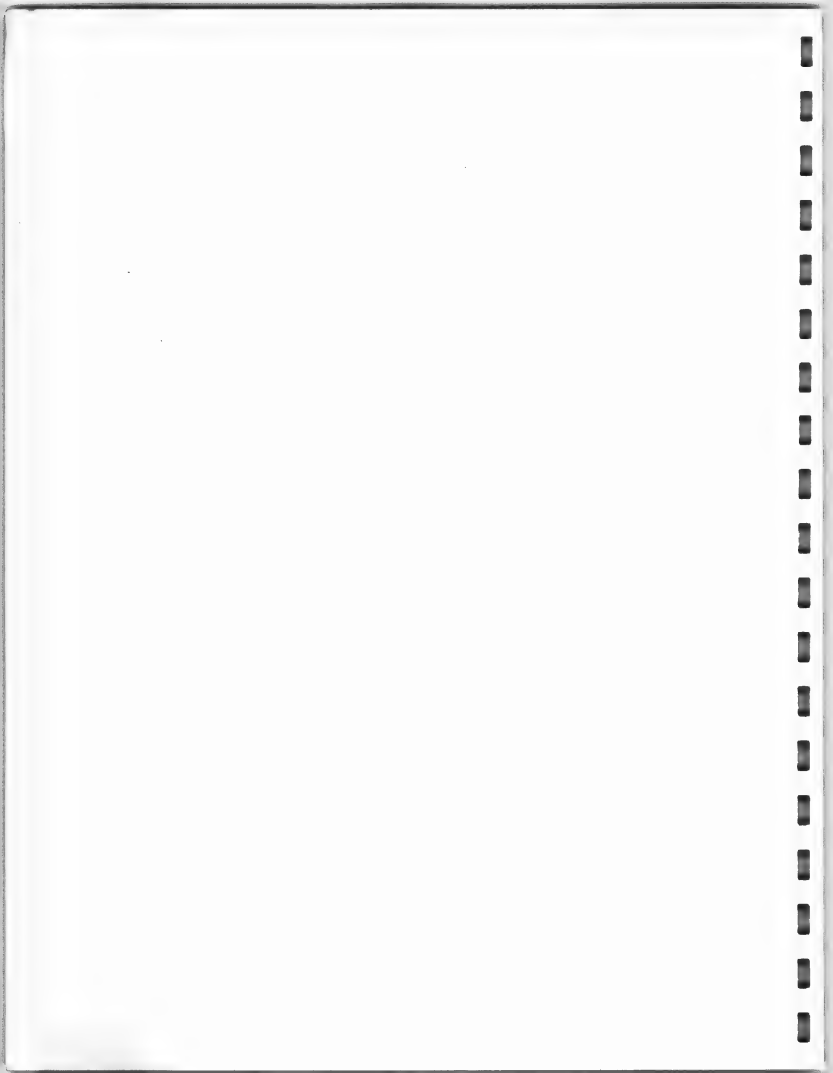
The literal signed, 8-bit value (in range +127 to -128) entered as 'LIT' is right justified, sign extended and loaded into 'REG T'. The condition code is set.

(REG T) ← 'LITERAL'
SET CCR

LOAD-REGISTER, REGISTER-INDEXEDMODE DESIGNATOR RXASSEMBLER FORMAT LDR RX, 'REG T', 'REG S', 'REG X'RX FORMATDESCRIPTION

The contents of 'REG X' are added to the contents of 'REG S' to form an address. The contents of that address are loaded into 'REG T', and the condition code is set.

$(\text{REG T}) + ((\text{REG X}) + (\text{REG S}))$
SET CCR

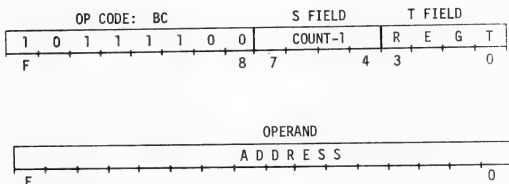


LDRM

LOAD MULTIPLE

MODES: D, DR

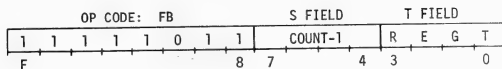
CONDITION CODE: The condition code is not
changed by execution of these
instructions.

LOAD MULTIPLE DIRECTMODE DESIGNATOR DASSEMBLER FORMAT LDRM D, 'REG T', 'ADR', 'COUNT'NL FORMATDESCRIPTION

The contents of memory locations beginning with 'ADDRESS' are loaded into consecutive registers beginning with 'REG T'.* 'COUNT' is the number of registers to be loaded. The condition code is not changed by this instruction.

(REG T) THRU (REG T + 'COUNT'-1)* LOADED FROM MEMORY LOCATIONS:
'ADDRESS' THRU 'ADDRESS'+ 'COUNT'-1

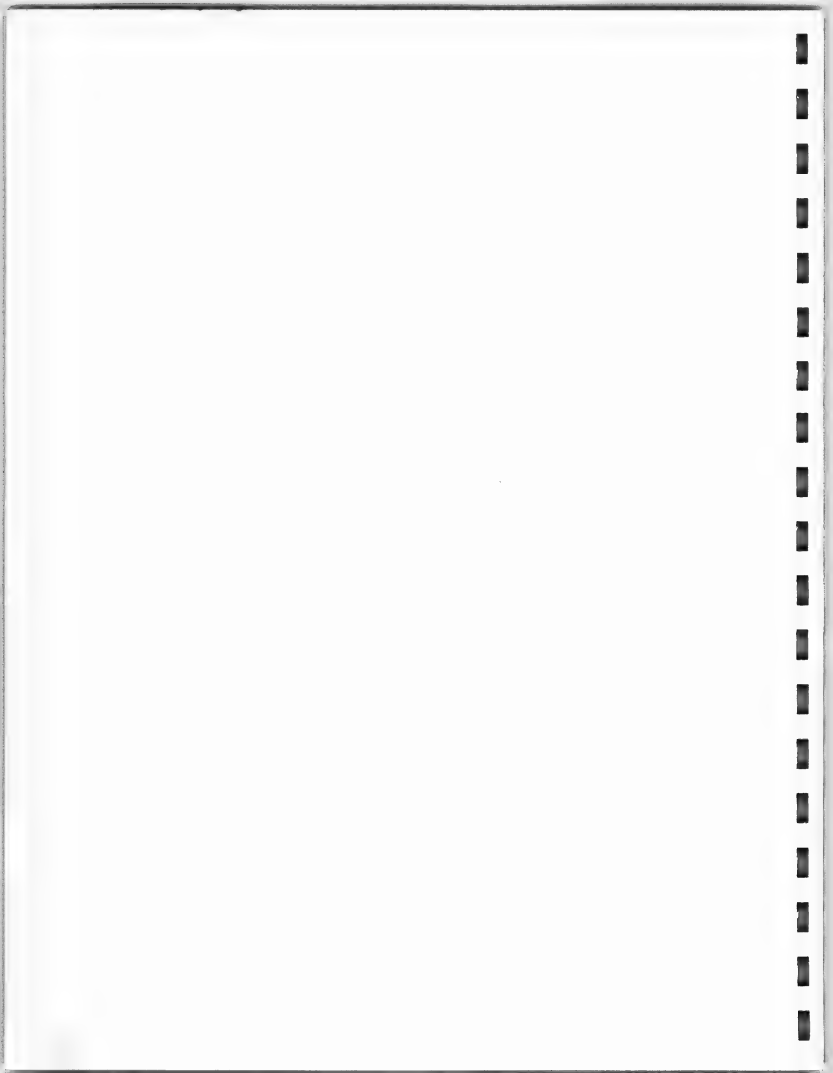
*MODULO 16

LOAD MULTIPLE, DIRECT REGISTERMODE DESIGNATOR DRASSEMBLER FORMAT LDRM DR, 'REG T', 'COUNT'RR FORMATDESCRIPTION

The contents of memory locations beginning with (REG T) are loaded into consecutive registers beginning with 'REG T' +1 and ending with REG T + 'COUNT'*. 'COUNT' is the number of registers to be loaded. The condition code is not changed by this instruction.

(REG T+1) THRU (REG T + 'COUNT')* LOADED FROM MEMORY LOCATIONS:
 (REG T) THRU (REG T)+'COUNT'-1

*MODULO 16



LOAD STATUS

MODES: D, DX

CONDITION CODE: The condition code, program counter and interrupt level are restored to a set of stored, previous values.

NOTE

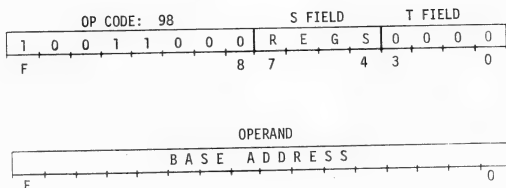
This instruction is designed to be executed at the exit from an interrupt service routine or from a routine called using the TRAP instruction (page 6-223). Execution of this instruction restores computer status which was automatically saved by ATAC when the interrupt/TRAP occurred.

DESCRIPTION (continued)

Software Testable CCR	Hardware CCR	Status Word CCR*
Ov N Z P	Z N O C**	Z N O C
0 0 0 1	0 0 0 X	1 1 1 \bar{X}
0 0 1 0	1 0 0 X	0 1 1 \bar{X}
0 1 0 0	0 1 0 X	1 0 1 \bar{X}
1 0 0 1	0 0 1 X	1 1 0 \bar{X}
1 0 1 0	1 0 1 X	0 1 0 \bar{X}
1 1 0 0	0 1 1 X	1 0 0 \bar{X}

* This version of the CCR is also exhibited on the minipanel as register 13₁₆.

** C = Carry indicator: set to 1 when arithmetic operands regarded as 16 (32)-bit unsigned quantities produce a 17 (33)-bit result.

LOAD STATUS, DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT LDST DX, 'BASE', 'REG S'NL FORMATDESCRIPTION

The address formed by adding the 'BASE ADDRESS' with 'REG S' points to a two-word save area containing two words: the interrupt priority level (IL) concatenated with the condition code register (CCR), and the program counter. Execution of this instruction causes these two words to be loaded as computer status. The 'S' and 'T' fields are not used.

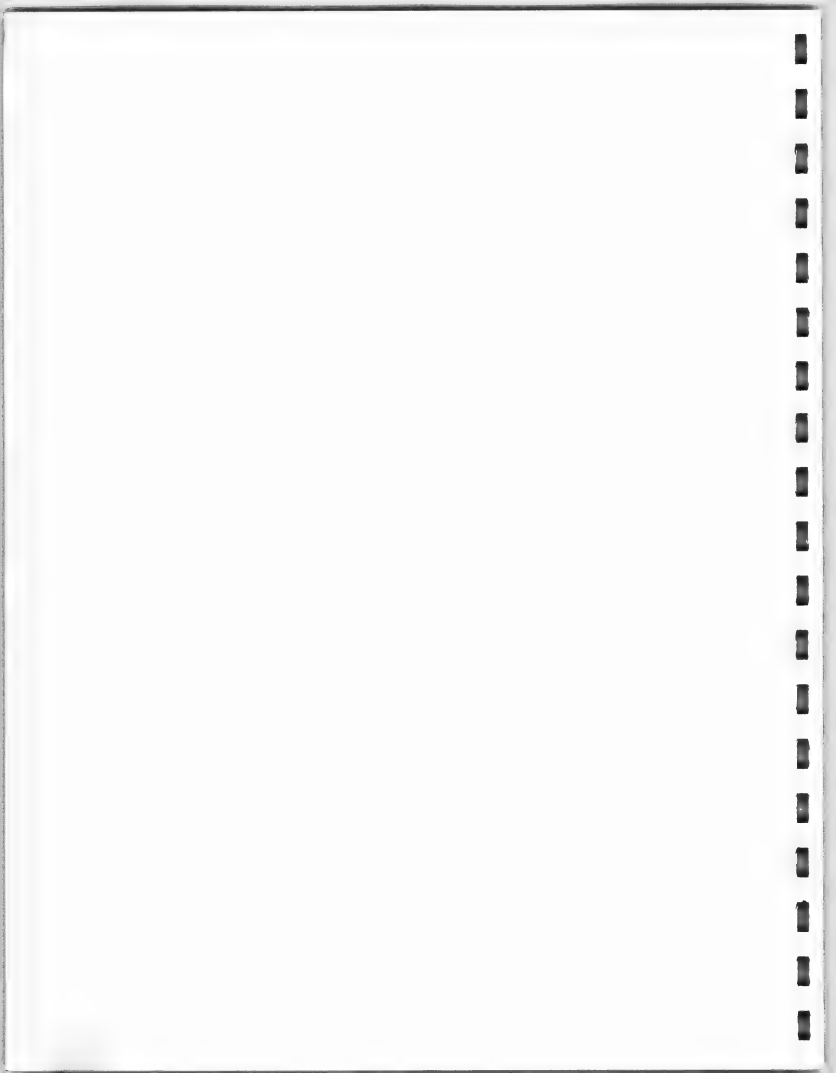
Notes: Interrupt level zero is the highest priority interrupt. When the interrupt status is restored to the ATAC-16M, only interrupts of the same or higher priority than the level restored (IL) will be allowed to interrupt the processor. The CCR consists of four bits of ALU status information as defined below. Refer also to section 3.2.

DESCRIPTION (Continued)

Software Testable CCR	Hardware CCR	Status Word CCR*
Ov N Z P	Z N O C**	$\bar{Z} \bar{N} \bar{O} \bar{C}$
0 0 0 1	0 0 0 X	1 1 1 \bar{X}
0 0 1 0	1 0 0 X	0 1 1 \bar{X}
0 1 0 0	0 1 0 X	1 0 1 \bar{X}
1 0 0 1	0 0 1 X	1 1 0 \bar{X}
1 0 1 0	1 0 1 X	0 1 0 \bar{X}
1 1 0 0	0 1 1 X	1 0 0 \bar{X}

* This version of the CCR is also exhibited on the minipanel as register 13₁₆.

** C = Carry indicator: set to 1 when arithmetic operands regarded as 16 (32)-bit unsigned quantities produce a 17 (33)-bit result.



LD4B

UB

LOAD UPPER BYTE

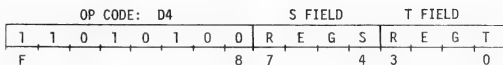
MODES: R, I, D, DX

CONDITION CODE: The condition code for 'LOAD
UPPER BYTE' is always set by
the final contents of 'REG T'
as follows:

(REG T) > 0, Condition code is 0001

(REG T) = 0, Condition Code is 0010

Condition Code 0100 is not used.

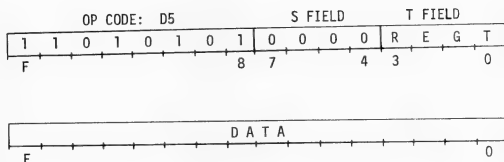
LOAD UPPER BYTE REGISTERMODE DESIGNATOR RASSEMBLER FORMAT LDUB R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The high order byte (bits F-8) of the contents of 'REG S' is loaded into the low order byte of 'REG T'. Bits F to 8 of 'REG T' are set to zero. The condition code is set.

$$(\text{REG T}, 7-0) \leftarrow (\text{REG S}, \text{F}-8)$$

$$(\text{REG T}, \text{F}-8) \leftarrow 0$$

SET CCR

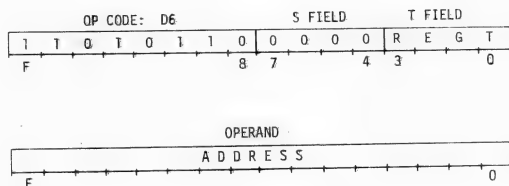
LOAD UPPER BYTE, IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT LDUB I, 'REG T', 'DATA'NL FORMATDESCRIPTION

The upper byte (bits F-8) of the value entered as 'DATA' is loaded as the low order byte of 'REG T'. Bits F to 8 of 'REG T' are set to zero. The condition code is set, and the 'S' field is not used.

(REG T, 7-0) ← 'DATA', BITS F-8

(REG T, F-8) ← 0

SET CCR

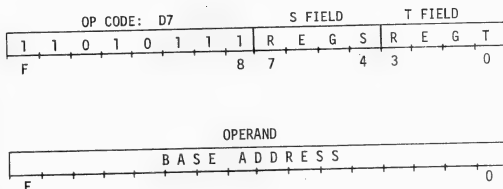
LOAD UPPER BYTE DIRECTMODE DESIGNATOR DASSEMBLER FORMAT LDUB D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The high order byte (bits F-8) of the contents of 'ADDRESS' is loaded into the low order byte (bits 7-0) of 'REG T'. Bits F-8 of 'REG T' are set to zero. The condition code is set. The 'S' field is not used.

(REG T, 7-0) ← (ADDRESS), BITS F-8

(REG T, F-8) ← 0

SET CCR

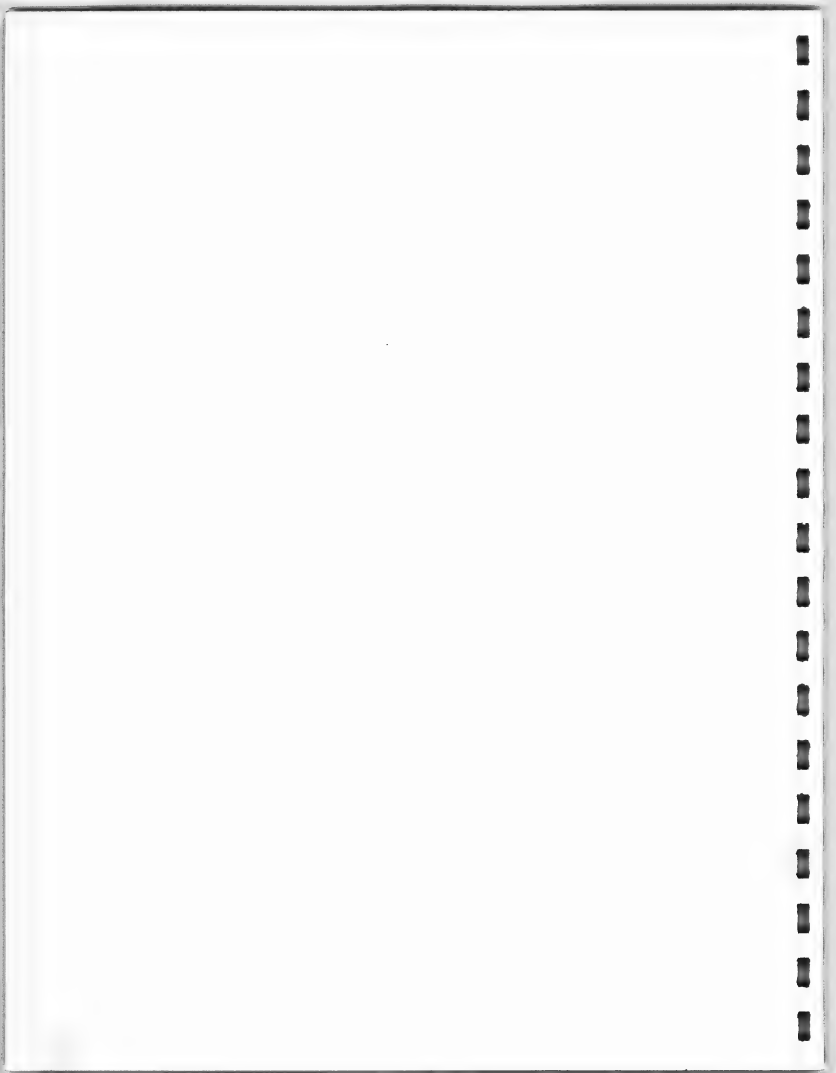
LOAD UPPER BYTE DIRECT INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT LDUB DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The high order byte (bits F-8) of the contents of the address formed by adding 'BASE' and 'REG S' is loaded into the low order byte (bits 7-0) of 'REG T'. Bits F to 8 of 'REG T' are set to zero. The condition code is set.

(REG T, 7-0) ← (BASE+(REGS)), BITS F-8

(REG T, F-8) ← 0

SET CCR

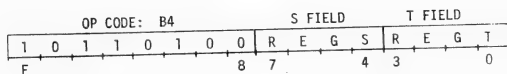


LOAD LOGICAL (ONE'S) COMPLEMENT

MODES: R, I, D, DX

CONDITION CODE: The condition code for 'LOAD
LOGICAL COMPLEMENT' instructions
is based on the final contents
of 'REG T' as follows:

(REG T) > 0, Condition code is 0001
(REG T) = 0, Condition code is 0010
(REG T) < 0, Condition code is 0100

LOAD LOGICAL (ONE'S) COMPLEMENT-REGISTERMODE DESIGNATOR RASSEMBLER FORMAT LLC R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The one's complement of the value in
'REG S' is loaded into 'REG T', and
the condition code is set.

$$(\text{REG T}) \leftarrow \overline{(\text{REG S})}$$

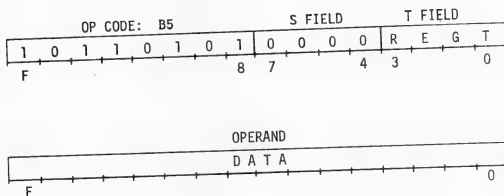
SET CCR

LOAD LOGICAL (ONE'S) COMPLEMENT IMMEDIATE

MODE DESIGNATOR I

ASSEMBLER FORMAT LLC I, 'REG T', 'DATA'

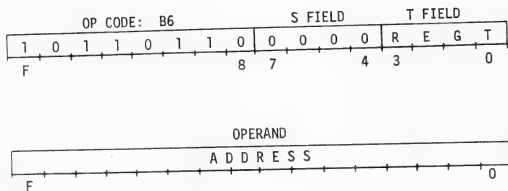
NL FORMAT



DESCRIPTION

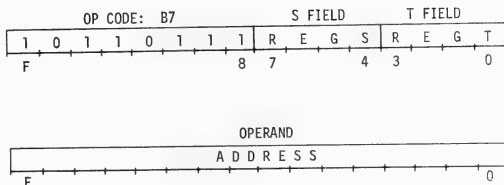
The one's complement of the value entered as 'DATA' is loaded into 'REG T'. The condition code is set. The 'S' field is not used.

(REG T) ← 'DATA'
SET CCR

LOAD LOGICAL (ONE'S) COMPLEMENT DIRECTMODE DESIGNATOR DASSEMBLER FORMAT LLC D, 'REG T', 'ADR'NL FORMATDESCRIPTION

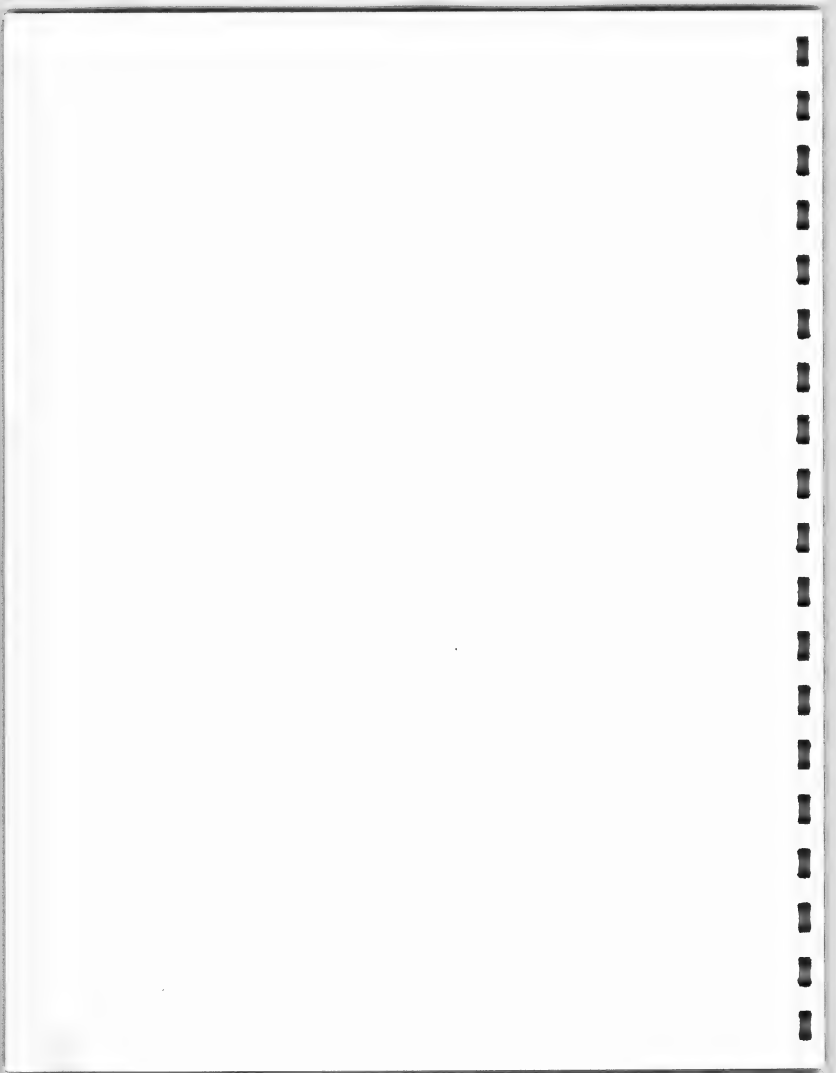
The one's complement of the contents of 'ADDRESS' is loaded into 'REG T', and the condition code is set. The 'S' field is not used.

(REG T) ← (ADDRESS)
SET CCR

LOAD LOGICAL (ONE'S) COMPLEMENT DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT LLC DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The one's complement of the content of the address formed by adding 'BASE' to 'REG S' is loaded into 'REG T'. The condition code is set.

$(\text{REG T}) \leftarrow \overline{(\text{BASE} + (\text{REG S}))}$
 SET CCR



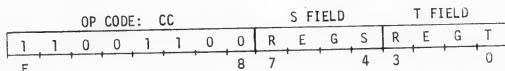
MUL

MULTIPLY

MODES: R, I, D, DX

CONDITION CODE: The condition code for
'MULTIPLICATION' instructions is
set by the contents of the 32 bit
product appearing in 'REG T+1',
'REG T' as follows:

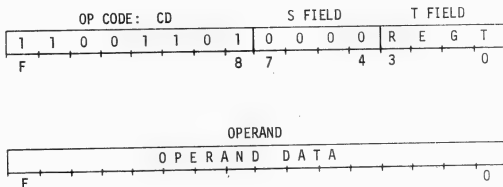
(REG T+1, T) > 0, Condition Code is 0001
(REG T+1, T) = 0, Condition Code is 0010
(REG T+1, T) < 0, Condition Code is 0100

MULTIPLY REGISTERMODE DESIGNATOR RASSEMBLER FORMAT MUL R, 'REG T', 'REG S'RR FORMATDESCRIPTION

The value in 'REG T' (multiplicand) is multiplied by the value in 'REG S' (multiplier). The product consists of 32 bits and appears in 'REG T+1' (most significant) and 'REG T' (least significant). Two's complement arithmetic is performed, and the condition code is set.

Note: This is an Integer Multiply; i.e., the effective position of the operand binary points is to the right of bit number 0.
Thus, $4000_{16} \times 4000_{16} \rightarrow 10000000_{16}$.

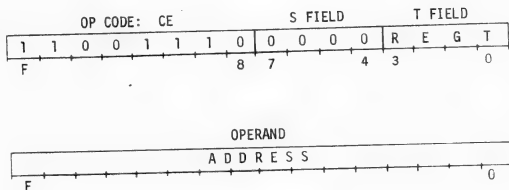
$(\text{REG T}+1) || (\text{REG T}) \leftarrow (\text{REG T}) * (\text{REG S})$
Set CCR

MULTIPLY IMMEDIATEMODE DESIGNATOR IASSEMBLER FORMAT MUL I, 'REG T', 'DATA'NL FORMATDESCRIPTION

The value in 'REG T' (multiplicand) is multiplied by the value entered as 'DATA' (multiplier). The 32-bit product appears in 'REG T' (least significant) and 'REG T+1' (most significant). The condition code is set; the 'S' field is not used.

Note: This is an Integer Multiply; i.e., the effective position of the operand binary points is to the right of bit number 0.
 Thus, $4000_{16} \times 4000_{16} \rightarrow 10000000_{16}$.

(REG T+1) || (REG T) ← (REG T) * 'DATA'
 Set CCR

MULTIPLY DIRECTMODE DESIGNATOR DASSEMBLER FORMAT MUL D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The value in 'REG T' (multiplicand) is multiplied by the value at 'ADDRESS'. The 32-bit product appears in 'REG T' (least significant half) and 'REG T+1' (most significant half). The condition code is set. The 'S' field is not used.

Note: This is an Integer Multiply; i.e., the effective position of the operand binary points is to the right of bit number 0.
Thus, $4000_{16} \times 4000_{16} \rightarrow 10000000_{16}$.

$(\text{REG T}+1) \parallel ((\text{REG T}) + (\text{REG T}) * (\text{ADDRESS}))$

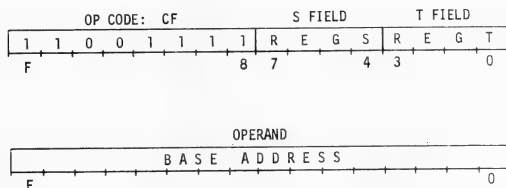
Set CCR

MULTIPLY DIRECT INDEX

MODE DESIGNATOR DX

ASSEMBLER FORMAT MUL DX, 'REG T', 'BASE', 'REG S'

NL FORMAT



DESCRIPTION

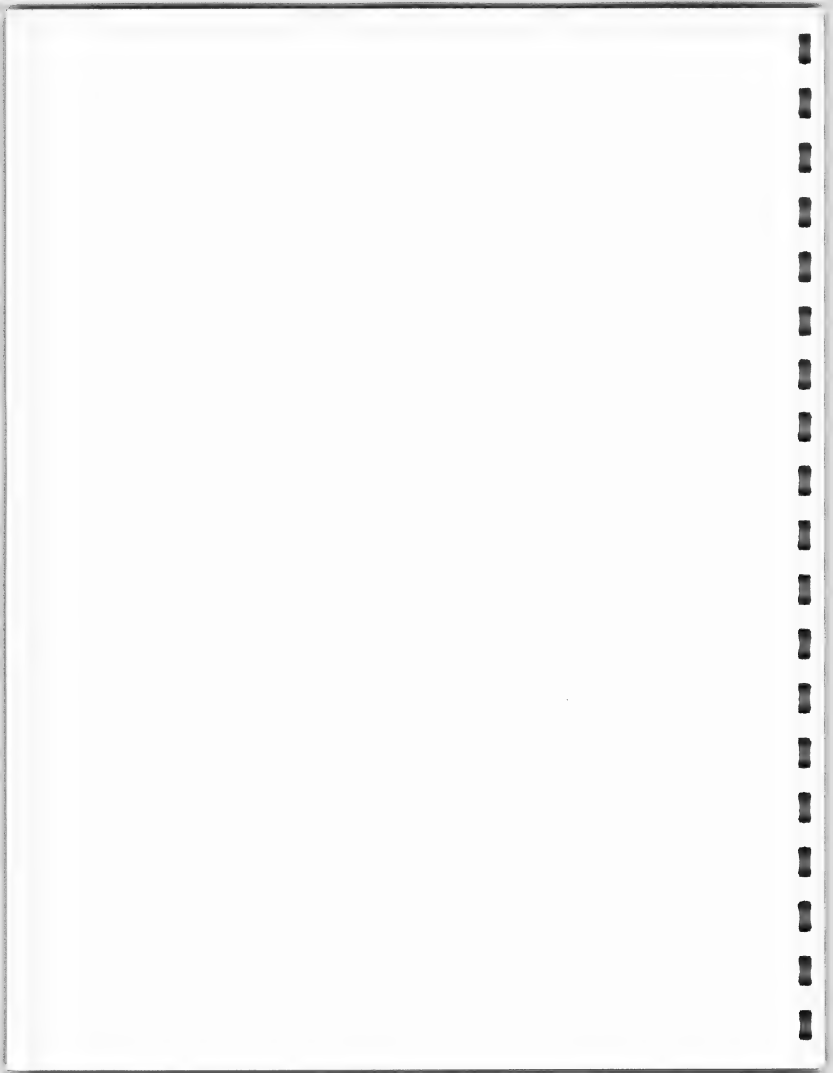
The value in 'REG T' (multiplicand) is multiplied by the value (multiplier) at the address formed by adding 'BASE ADDRESS' to the index in 'REG S'. THE 32-bit product appears in 'REG T' (least significant half) and 'REG T+1' (most significant half). The condition code is set.

Note: This is an Integer Multiply; i.e., the effective position of the operand binary points is to the right of bit number 0.
Thus, $4000_{16} \times 4000_{16} \rightarrow 10000000_{16}$.

$(\text{REG T}+1) || (\text{REG T}) \leftarrow (\text{REG T}) * (\text{BASE} + (\text{REG S}))$

Set CCR

6-159(6-160 blank)

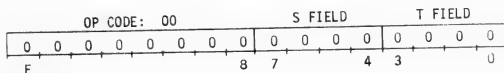


NOP

NO OPERATION

MODES: NO MODES

CONDITION CODE: The condition code is not changed
by this instruction.

NO OPERATIONMODE DESIGNATOR NO MODESASSEMBLER FORMAT NOPRR FORMATDESCRIPTION

The 'NOP' instruction causes no change in the computer status, except the normal increment of the program counter.

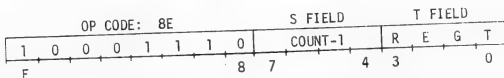
Note: This is a BRC IS instruction with a null (no branch) mask. Bits 4 through 8 are "don't care".

RETURN

RET

MODES: DR

CONDITION CODE: The condition code is not changed
by this instruction.

RETURNMODE DESIGNATOR DRASSEMBLER FORMAT RET DR, 'REG T', 'COUNT'RR FORMATDESCRIPTION

This instruction restores registers which were previously stacked by the 'STACK' instruction. 'REG T' acts as a stack pointer; its contents always address the next free stack location. After execution of the instruction, the stack pointer is decremented by the 'COUNT' value. It is the programmer's responsibility to define, initialize, and maintain any of the 16 general registers as stack pointer register(s). 'REG T' contains the stack pointer, and 'COUNT' is the number of registers to be returned. Starting with the register 'REG T' + 'COUNT', the 'COUNT' registers are restored in descending order from memory locations beginning at ((REG T)-1). (REG T) is decremented by 'COUNT'.

(REG T+COUNT)* THRU (REG T + 1)* restored from memory locations:
 (REG T) -1 THRU (REG T)-'COUNT'
 (REG T) + (REG T)-'COUNT'

*Modulo 16

REGISTER INPUT INSTRUCTION

MODES: NO MODES

RIN

CONDITION CODE: The condition code is set by the
value read into 'REG T' as follows:

(REG T) > 0, Condition Code is 0001

(REG T) = 0, Condition Code is 0010

(REG T) < 0, Condition Code is 0100

If the interface times out, the condition
code is set to 1100.

REGISTER INPUTMODE DESIGNATOR NO MODESASSEMBLER FORMAT RIN 'REG T', 'REG S'RR FORMATDESCRIPTION

The contents of 'REG S' form a 16-bit device address which defines the input device. A 16-bit parallel data input from the device is stored in 'REG T'. If an addressed I/O device fails to respond within 64 microseconds, the overflow is set, and the next sequential instruction is executed.

REGISTER OUTPUT INSTRUCTION

MODES: NO MODES

CONDITION CODE: The condition code is set by the value to be output from 'REG T' as follows:

ROUT

(REG T) > 0, Condition Code is 0001

(REG T) = 0, Condition Code is 0010

(REG T) < 0, Condition Code is 0100

If the interface subsequently times out, the condition code is overflow complemented as follows:

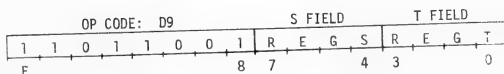
CONDITION CODE	
WAS	BECOMES
0 0 0 1	1 1 0 0
0 0 1 0	1 0 1 0
0 1 0 0	1 0 0 1

REGISTER OUTPUTMODE DESIGNATOR

NO MODES

ASSEMBLER FORMAT

ROUT 'REG T', 'REG S'

RR FORMATDESCRIPTION

The contents of 'REG S' form a 16-bit device address which defines the output device. The 16-bit parallel data value in 'REG T' is transferred to the addressed device. If the addressed I/O device fails to respond within 64 microseconds, the overflow is set and the next sequential instruction is executed.

SEARCH LOWER BYTE, EVERY NTH WORD

MODES: AL,BL

CONDITION CODE: Upon completion of the search instruction, the condition code is set as follows:

CCR=0010(ZERO) SUCCESS, Table entry equals search value

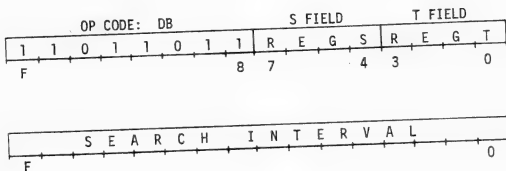
CCR=0100(NEG) SUCCESS, Table entry does not equal search value

CCR=0001(POS) FAILURE, Table search completed

SCHL

NOTE

1. These instructions are interruptable after each table entry is searched. After interrupt service, the search resumes at the next table search location.
2. If the search is successful, the address of the successful value is: Table end address + index, which can be referenced by an 'RX' type load.
3. If the search is successful, re-executing the instruction will start with the last table entry that satisfied the search criterion.
4. Byte values are assumed to be signed quantities.

SEARCH LOWER BYTE, EVERY NTH WORD; ABOVE OR EQUAL TO A LIMIT VALUEMODE DESIGNATOR ALASSEMBLER FORMAT SCHL AL, 'REG T', 'SEARCH INTERVAL', 'REG S'NL FORMATDESCRIPTION

'REG S' contains the address of the last word in the table to be searched.

'REG S + 1' contains the byte search limit value for comparison with search table entries. This limit must be left justified in bits F through 8, bits 7 through 0 must be zero filled.

'REG T' contains the index to the search table relative to the table end address. This value is initially defined as the difference between the addresses of the start and end of the table. Upon exit from a successful search instruction, 'REG T' contains the address of the successful table entry with respect to the table end address.

DESCRIPTION (continued)

'SEARCH INTERVAL' is the increment between values addressed in the table.

The address of the table entry is defined as the sum of the table end address (REG S) plus the index (REG T). The byte search treats byte values as signed 8-bit integers.

(REG S): TABLE END ADDRESS

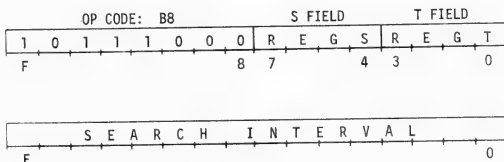
(REG S + 1): BITS F-8 + LIMIT BYTE VALUE; BITS
7-0 + ZERO

(REG T): INDEX TO TABLE (TABLE ENTRY ADDRESS-TABLE
END ADDRESS)

TABLE VALUE ADDRESS = (REG S) + (REG T)

The search instruction iterates as follows:

1. The table entry address is computed;
2. The lower byte of the addressed table value is compared to the upper byte of the limit value (REG S + 1);
3. If the lower byte of the table entry is greater than or equal to the upper byte of the limit value, the search is terminated with the CCR set to "SUCCESS" (equal: CCR=0010) (above: CCR=0100);
4. If the comparison fails, the SEARCH INTERVAL is added to the index (REG T);
5. If the index is still negative, continue the search at step 1 (above).
6. A positive index value terminates the search instruction with the CCR set to "FAIL" (CCR=0001).

SEARCH LOWER BYTE EVERY NTH WORD; BELOW OR EQUAL TO A LIMIT VALUEMODE DESIGNATOR BLASSEMBLER FORMAT SCHL BL, 'REG T', 'SEARCH INTERVAL', 'REG S'NL FORMATDESCRIPTION

'REG S' contains the address of the last word in the table to be searched.

'REG S + 1' contains the byte search limit value for comparison with search table entries. This limit must be left justified in bits F through 8, bits 7 through 0 must be one-filled ('FF').

'REG T' contains the index to the search table relative to the table end address. This value is initially defined as the difference between the addresses of the start and end of the table. Upon exit from a successful search instruction, 'REG T' contains the address of the successful table entry with respect to the table end address.

DESCRIPTION (continued)

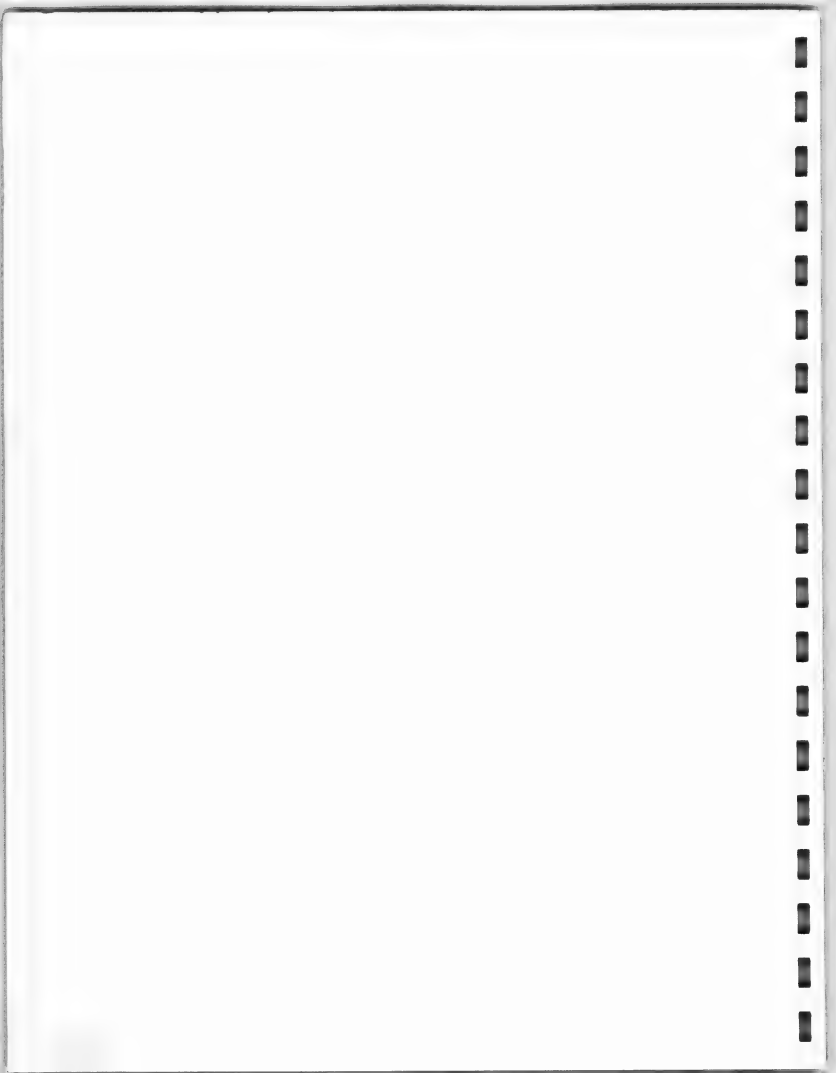
'SEARCH INTERVAL' is the increment between values addressed in the table.

The address of the table entry is defined as the sum of the table end address (REG S) plus the index (REG T). The byte search treats byte values as signed 8-bit integers.

(REG S): TABLE END ADDRESS
 (REG S + 1): BITS F-8 + LIMIT BYTE VALUE; BITS
 7-0 + 'FF'
 (REG T): INDEX TO TABLE (TABLE ENTRY ADDRESS-TABLE
 END ADDRESS)
 TABLE VALUE ADDRESS = (REG S) + (REG T)

The search instruction iterates as follows:

1. The table entry address is computed;
2. The lower byte of the addressed table value is compared to the upper byte of the limit value (REG S + 1);
3. If the lower byte of the table entry is less than or equal to the upper byte of the limit value, the search is terminated with the CCR set to "SUCCESS" (equal: CCR=0010) (below: CCR=0100);
4. If the comparison fails, the SEARCH INTERVAL is added to the index (REG T);
5. If the index is still negative, continue the search at step 1 (above).
6. A positive index value terminates the search instruction with the CCR set to "FAIL" (CCR=0001).



SEARCH EVERY NTH WORD

MODES: AL,BL

CONDITION CODE: Upon completion of the search instruction, the condition code is set as follows:

CCR=0010(ZERO) SUCCESS, Table entry equals search value

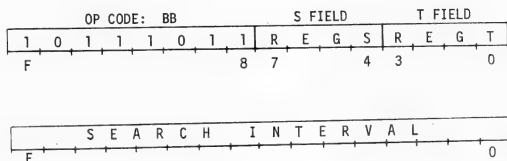
CCR=0100(NEG) SUCCESS, Table entry does not equal search value

CCR=0001(POS) FAILURE, Table search completed

NOTE

1. These instructions are interruptable after each table entry is searched. After interrupt service, the search resumes at the next table search location.
2. If the search is successful, the address of the successful value is: TABLE END ADDRESS + INDEX which can be referenced by an 'RX' type load.
3. If the search is successful, re-executing this instruction will start with the last table entry that satisfied the search criterion.
4. This instruction may be used to search signed upper bytes of a word table. The byte limit must be left-justified, and the low order byte of the limit word filled with 'FF' for the below limit search, or '00' for the above limit search. In general, CCR is not set to (0010) if byte equality exists between the table value and limit.

SCHW

SEARCH EVERY NTH WORD ABOVE OR EQUAL TO LIMITMODE DESIGNATOR ALASSEMBLER FORMAT SCHW AL, 'REG T', 'SEARCH INTERVAL', 'REG S'NL FORMATDESCRIPTION

'REG S' contains the address of the last word in the table to be searched.

'REG S + 1' contains the word search limit value for comparison with search table entries. This limit is a 2's complement 16-bit number.

'REG T' contains the index to the search table relative to the table end address. This value is initially defined as the difference between the addresses of the start and end of the table. Upon exit from a successful search instruction, 'REG T' contains the address of the successful table entry with respect to the table end address.

DESCRIPTION (continued)

'SEARCH INTERVAL' is the increment between values addressed in the table.

The address of the table entry is defined as the sum of the table end address (REG S) plus the index (REG T). The byte search treats byte values as positive integers.

(REG S): TABLE END ADDRESS
(REG S + 1): LIMIT VALUE
(REG T): INDEX TO TABLE (TABLE ENTRY ADDRESS-TABLE
 END ADDRESS)

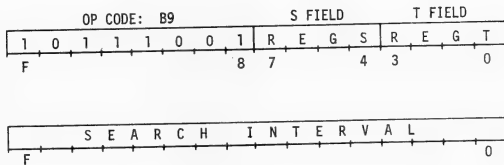
TABLE VALUE ADDRESS = (REG S) + (REG T)

The search instruction iterates as follows:

1. The table entry address is computed;
2. The addressed table value is compared to the limit value (REG S + 1);
3. If the table entry is greater than or equal to the limit value, the search is terminated with the CCR set to "SUCCESS" (equal: CCR=0010) (above: CCR=0100);
4. If the comparison fails, the SEARCH INTERVAL is added to the index ('REG T');

DESCRIPTION (continued)

5. If the index is still negative, continue the search at step 1 (above).
6. A positive index value terminates the search instruction with the CCR set to "FAIL" (CCR=0001).

SEARCH EVERY NTH WORD BELOW OR EQUAL TO LIMITMODE DESIGNATOR BLASSEMBLER FORMAT SCHW BL, 'REG T', 'SEARCH INTERVAL', 'REG S'NL FORMATDESCRIPTION

'REG S' contains the address of the last word in the table to be searched.

'REG S + 1' contains the word search limit value for comparison with search table entries. This limit is a 2's complement 16-bit number.

'REG T' contains the index to the search table relative to the table end address. This value is initially defined as the difference between the addresses of the start and end of the table. Upon exit from a successful search instruction, 'REG T' contains the address of the successful table entry with respect to the table end address.

DESCRIPTION (Continued)

'SEARCH INTERVAL' is the increment between values addressed in the table.

The address of the table entry is defined as the sum of the table end address (REG S) plus the index (REG T). The byte search treats byte values as signed 8-bit integers.

(REG S): TABLE END ADDRESS

(REG S + 1): LIMIT VALUE

(REG T): INDEX TO TABLE (TABLE ENTRY ADDRESS-TABLE
 END ADDRESS)

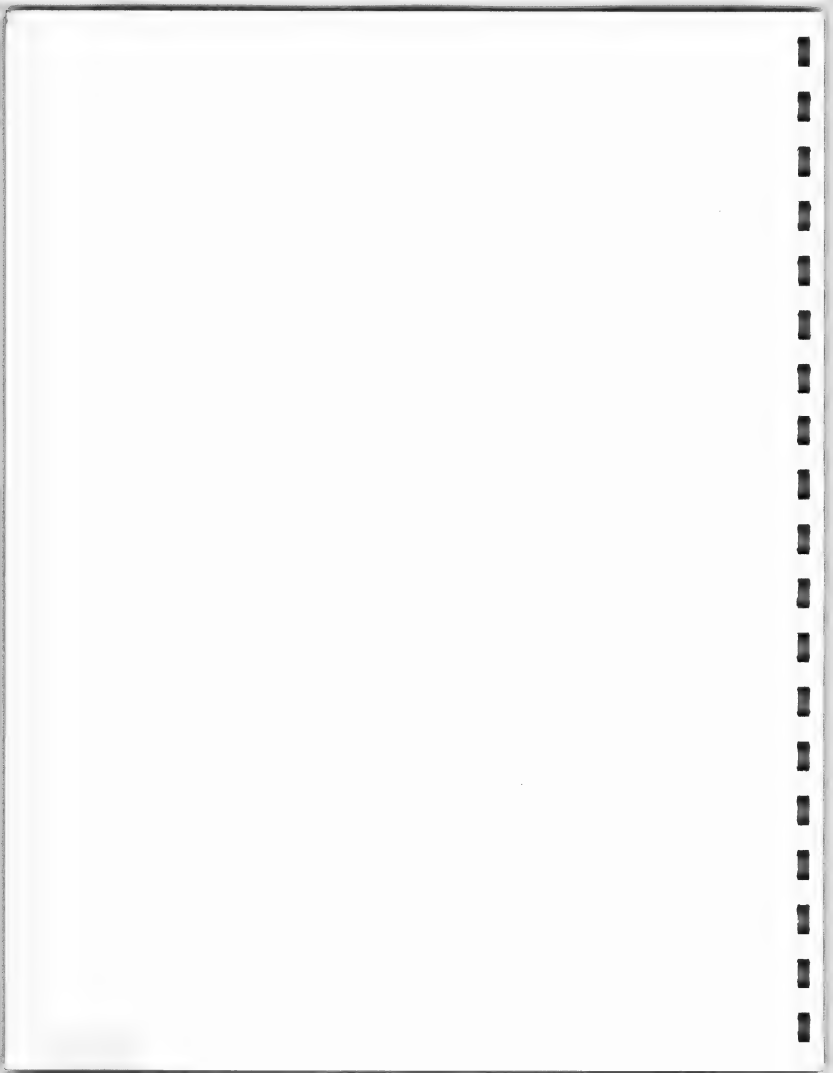
TABLE VALUE ADDRESS = (REG S) + (REG T)

The search instruction iterates as follows:

1. The table entry address is computed;
2. The addressed table value is compared to the limit value (REG S + 1);
3. If the table entry is less than or equal to the limit value, the search is terminated with the CCR set to "SUCCESS" (equal: CCR=0010) (below: CCR=0100);
4. If the comparison fails, the SEARCH INTERVAL is added to the index (REG T);

DESCRIPTION (Continued)

5. If the index is still negative, continue the search at step 1 (above).
6. A positive index value terminates the search instruction with the CCR set to "FAIL" (CCR=0001).



SHIFT DOUBLE REGISTER

MODES: LL, RA, RL

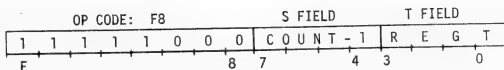
CONDITION CODE: The condition code is set by the final contents of 'REG T' + 1, 'T' as follows:

(REG T+1, T) > 0, Condition Code is 0001

(REG T+1, T) = 0, Condition Code is 0010

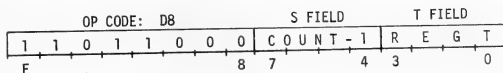
(REG T+1, T) < 0, Condition Code is 0100

SHD

SHIFT DOUBLE REGISTER LEFT LOGICALMODE DESIGNATOR LLASSEMBLER FORMAT SHD LL, 'REG T', 'COUNT'RR FORMATDESCRIPTION

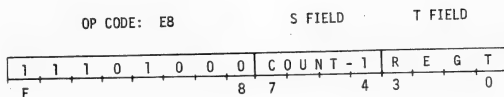
The contents of 'REG T+1' and 'REG T' are treated as a single 32 bit register and shifted left the number of bits specified by 'COUNT'. The 'COUNT' value range is 1 to 16. Right fill with zero in 'REG T', and set the condition code.

$(\text{REG T}+1 \parallel \text{REG T}) \leftarrow \text{SHIFTED LEFT BY 'COUNT' BITS}$
Set CCR

SHIFT DOUBLE REGISTER RIGHT ARITHMETICMODE DESIGNATOR RAASSEMBLER FORMAT SHD RA, 'REG T', 'COUNT'RR FORMATDESCRIPTION

The contents of 'REG T+1' and 'REG T' are treated as a single long register and shifted right the number of bits specified by 'COUNT'. The 'COUNT' value range is 1 to 16. Left fill with the sign bit of 'REG T+1', and the condition code is set.

(REG T+1||REG T) ← SHIFTED RIGHT BY 'COUNT' BITS
Set CCR

SHIFT DOUBLE REGISTER RIGHT LOGICALMODE DESIGNATOR RLASSEMBLER FORMAT SHD RL, 'REG T', 'COUNT'RR FORMATDESCRIPTION

The contents of 'REG T+1' and 'REG T' are treated as a single long register and shifted right 'COUNT' bits. The 'COUNT' value range is 1 to 16. Left fill with zeroes and set the condition code.

(REG T+1||REG T) ← SHIFTED RIGHT 'COUNT' BITS
Set CCR

SHIFT SINGLE REGISTER

MODES: LL, RA, RL

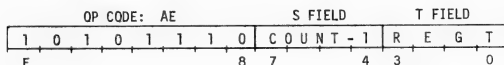
CONDITION CODE: The condition code for all shift
single register instructions is
based on the final contents of
'REG T' as follows:

(REG T) > 0, Condition Code is 0001

(REG T) = 0, Condition Code is 0010

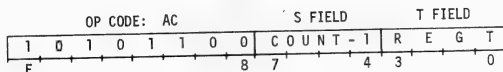
(REG T) < 0, Condition Code is 0100

SHS

SHIFT SINGLE REGISTER LEFT LOGICALMODE DESIGNATOR LLASSEMBLER FORMAT SHS LL, 'REG T', 'COUNT'RR FORMATDESCRIPTION

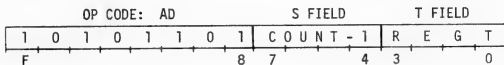
The contents of 'REG T' are shifted left the number bits indicated by the value 'COUNT'. The 'COUNT' value range is 1 to 16. Right fill with zeroes, and set the condition code.

(REG T) ← SHIFTED LEFT BY 'COUNT' BITS
Set CCR

SHIFT SINGLE REGISTER RIGHT ARITHMETICMODE DESIGNATOR RAASSEMBLER FORMAT SHS RA, 'REG T', 'COUNT'RR FORMATDESCRIPTION

The contents of 'REG T' are shifted right the number of bits indicated by the value 'COUNT'. The 'COUNT' value range is 1 to 16. Left fill with the sign bit, and the condition code is set.

(REG T) ← SHIFTED RIGHT BY 'COUNT' BITS
Set CCR

SHIFT SINGLE REGISTER RIGHT LOGICALMODE DESIGNATOR RLASSEMBLER FORMAT SHS RL, 'REG T', 'COUNT'RR FORMATDESCRIPTION

The contents of 'REG T' are shifted right the number of bits indicated by the value 'COUNT'. The 'COUNT' value range is 1 to 16. Left fill with zeroes, and the condition code is set.

(REG T) ← SHIFTED RIGHT BY 'COUNT' BITS
Set CCR

ATAC SIMULATOR CONTROL INSTRUCTION

MODES: NONE

CONDITION CODE: The 'SIM' instruction does not
change the condition code.

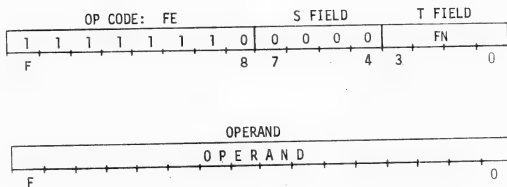
SIM

INSTRUCTION SIMULATOR CONTROL

MODE DESIGNATOR None

ASSEMBLER FORMAT SIM 'FN', 'OPERAND'

NL FORMAT



DESCRIPTION

ATAC treats all forms of the instruction simulator control instruction as a no-operation. This two-word instruction does not change the state of the machine, except to advance the program counter to execution of the next sequential instruction. Usage of this instruction for the control of simulation runs is discussed in the ATAC Volume 2 Programming Support System (APSS).

SET MEMORY SEMAPHORE

MODES: D, DX

CONDITION CODE: The condition code is set
based upon the success of
the SMS to set bit(s) in
memory.

Memory bit(s) set, condition code is 0010

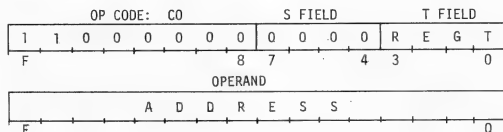
Memory bit(s) not set, condition code is 0001 or 0100

SMS

SET MEMORY SEMAPHORE DIRECT

MODE DESIGNATOR D

ASSEMBLER FORMAT SMS D, 'REG T', 'ADR'

NL FORMATDESCRIPTION

The SMS instruction provides a general purpose mechanism for processor capture of computer resources in multiprocessor configurations. Conversely, the CMS instruction is used to release the resource for use by another processor.

The SMS instruction utilizes 'REGT' as a mask so that any part of the 16-bit semaphore (located at 'ADDRESS' in memory) may be used as a flag to indicate capture of a resource. The SMS instruction ANDs the semaphore with (REGT) and successfully captures the resource when the condition code resulting from this AND indicates a zero result.

When a capture is successful, 'REGT' is inclusive ORed with the semaphore word and the result is rewritten to memory. In this manner all fTag bits are set, indicating to all external processors that the device has been captured.

DESCRIPTION (Continued)

The semaphore instruction locks out all other processor and I/O memory accesses for four microcycles (two with memory references). During the lockout the semaphore word is read from memory, is ANDed with 'REGT', and assuming capture of the resource, is inclusive ORed with 'REGT' and the result written to memory.

To prevent memory from being locked for an excessive amount of time while waiting for the resource to be released, the SMS instruction first reads and tests the semaphore word without the lockout. If the capture is unsuccessful, the instruction terminates with the CCR indicating a non-zero result. If the capture is successful with the memory lock off, the integrity of the capture is insured by re-reading and retesting the semaphore word with the lock on.

If the final test determines that the resource is available (i.e., bit not set) the instruction sets the semaphore flag bit(s) corresponding to any '1's in 'REGT' and rewrites the semaphore word to memory.

If the final test indicates that another processor has captured the resource since the previous test, the memory lockout is cleared and the instruction terminates.

Reading and testing the semaphore word first with the memory lock off and then (if the first test was successful) with the memory lock on allows for more efficient multiport access to the memory space containing the semaphore word.

The suggested method for using the SMS is as follows:

DESCRIPTION (Continued)

	LDR	I,2,0101	DATA FOR SEMAPHORE
WAITL	SMS	D,2,MEMSEM	CHECK SEMAPHORE
	BRCS	NE,WAITL	LOOP IF UNSUCCESSFUL
	.	.	
	.	.	CODE USING RESOURCE
	.	.	
	CMS	D,2,MEMSEM	RELEASE RESOURCE

<u>Step</u>	<u>Operation</u>	<u>Memory Lock</u>
1	ADR → memory address register, read next instruction.	no
2	Read semaphore word from memory.	no
3	AND semaphore word with REGT; set CCR on result.	no
4	Branch to step 9 if CCR does not indicate a zero result.	no
5	Reread semaphore word.	yes
6	AND semaphore word with REGT; set CCR on result.	yes
7	Branch to step 11 if CCR does not indicate a zero result; OR semaphore word with REGT, transfer result to memory output register.	yes
8	Write new semaphore word to memory.	yes
9	Increment program counter.	no
10	Perform an instruction fetch and exit.	no
11	Clear lockout by reading semaphore word again; increment program counter and branch to step 10.	no

DESCRIPTION (Continued)

The semaphore instruction locks out all other processor and I/O memory accesses for four microcycles (two with memory references). During the lockout the semaphore word is read from memory, is ANDed with 'REGT', and assuming capture of the resource, is inclusive ORED with 'REGT' and the result written to memory.

To prevent memory from being locked for an excessive amount of time while waiting for the resource to be released, the SMS instruction first reads and tests the semaphore word without the lockout. If the capture is unsuccessful, the instruction terminates with the CCR indicating a non-zero result. If the capture is successful with the memory lock off, the integrity of the capture is insured by re-reading and retesting the semaphore word with the lock on.

If the final test determines that the resource is available (i.e., bit not set) the instruction sets the semaphore flag bit(s) corresponding to any '1's in 'REGT' and rewrites the semaphore word to memory.

If the final test indicates that another processor has captured the resource since the previous test, the memory lockout is cleared and the instruction terminates.

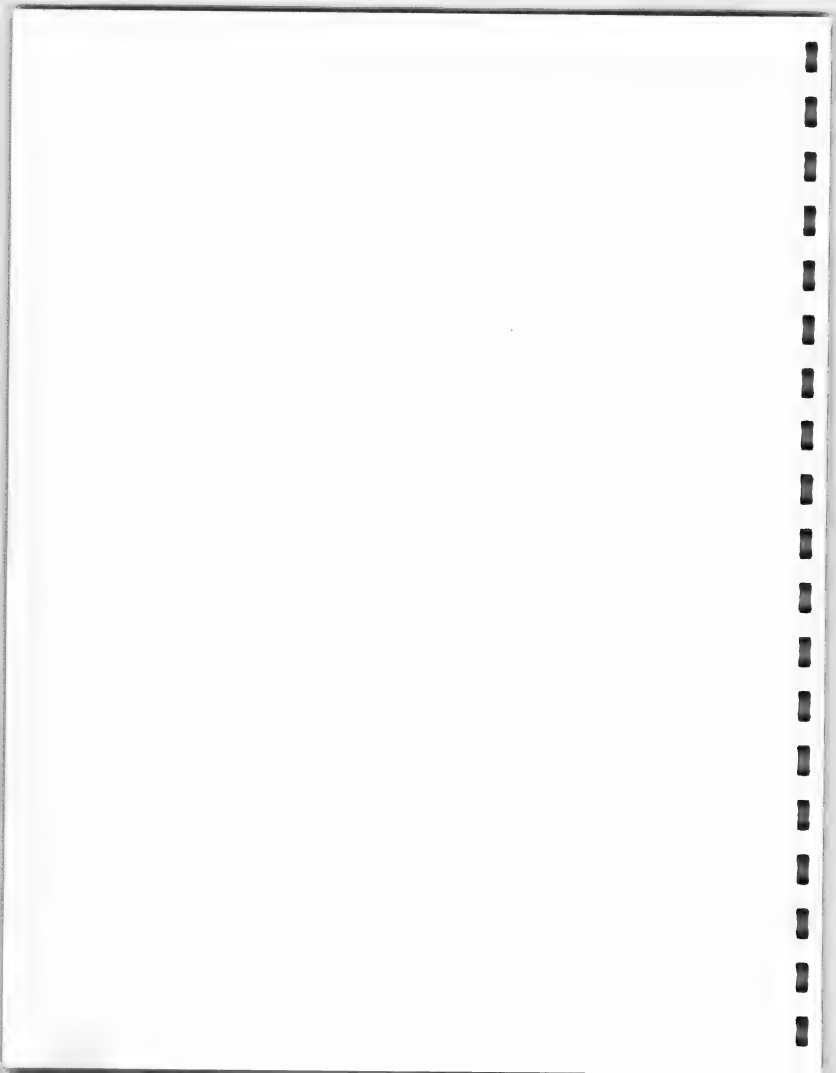
Reading and testing the semaphore word first with the memory lock off and then (if the first test was successful) with the memory lock on allows for more efficient multipoint access to the memory space containing the semaphore word.

The suggested method for using the SMS is as follows:

DESCRIPTION (Continued)

	CDR	I, 1, OFFSET	
	LDR	I, 2, 0101	DATA FOR SEMAPHORE
WAITL	SMS	DX, 2, SEMBASE, 1	CHECK SEMAPHORE
	BRCS	NE, WAITL	LOOP IF UNSUCCESSFUL
	.	.	
	.	.	CODE USING RESOURCE
	.	.	
	CMS	DX, 2, SEMBASE, 1	RELEASE RESOURCE

<u>Step</u>	<u>Operation</u>	<u>Memory Lock</u>
1	'BASE' + (REG S) memory address register, read next instruction.	no
2	Read semaphore word from memory.	no
3	AND semaphore word with REGT; set CCR on result.	no
4	Branch to step 9 if CCR does not indicate a zero result.	no
5	Reread semaphore word.	yes
6	AND semaphore word with REGT; set CCR on result.	yes
7	Branch to step 11 if CCR does not indicate a zero result; OR semaphore word with REGT, transfer result to memory output register.	yes
8	Write new semaphore word to memory.	yes
9	Increment program counter.	no
10	Perform an instruction fetch and exit.	no
11	Clear lockout by reading semaphore word again; increment program counter and branch to step 10.	no

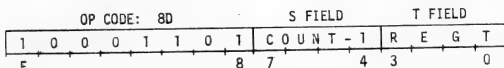


STACK

MODES: DR

CONDITION CODE: The condition code is not changed
by this instruction

STK

STACKMODE DESIGNATOR DRASSEMBLER FORMAT STK DR, 'REG T', 'COUNT'RR FORMATDESCRIPTION

When one or more registers are to be saved on a stack, the 'STACK' instruction is used. The programmer is responsible for defining, initializing, and maintaining any of the 16 general registers as stack pointer registers. A stack pointer register contains the address of the next memory location available for register storage. The 'RETURN' instruction is used to return register values from the stack. For every stack command executed, an equal number of registers stacked should be returned to registers from the stack, using the 'RETURN' instruction.

'REG T' points to the next available memory location for stacking. 'COUNT' is the number of registers to be stacked. Upon execution, contents of registers 'REG T' -1* thru

*Modulo 16

DESCRIPTION (continued)

'REG T' + 'COUNT'* are stored in ascending memory locations (REG T) through (REG T) + 'COUNT' -1. Upon completion of this instruction, 'REGT' is incremented to point to the next available stack location.

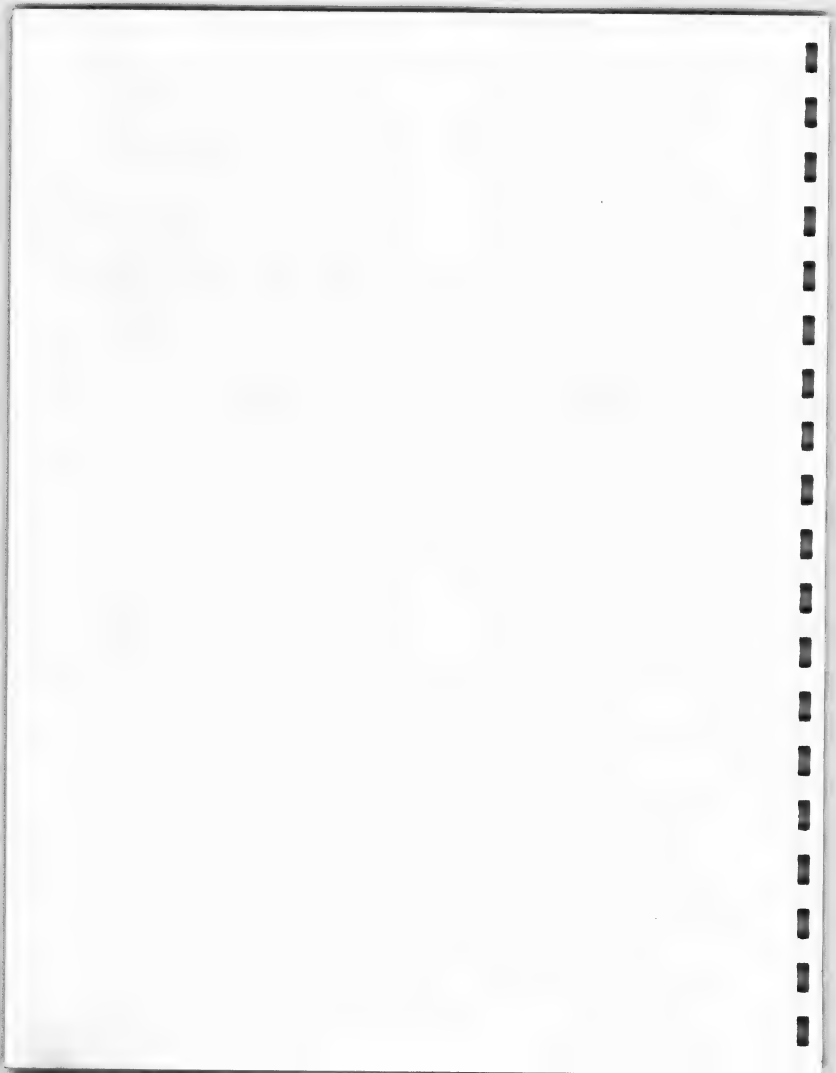
(REG T +1)* THRU (REG T + 'COUNT')* STORED IN MEMORY

LOCATIONS:

(REG T) THRU (REG T) + 'COUNT' -1;

(REG T) + (REG T) + 'COUNT'

*Modulo 16

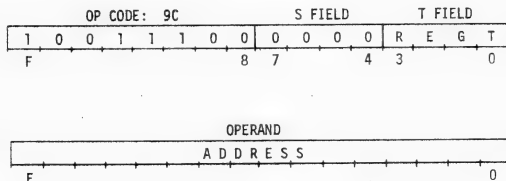


STORE

MODES: D, DX, RX

CONDITION CODE: The 'STORE' instructions leave
the condition code unchanged.

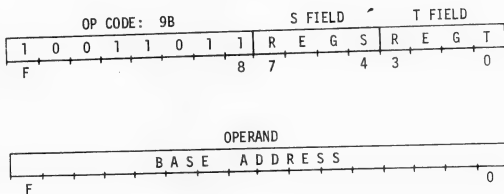
STR

STORE DIRECTMODE DESIGNATOR DASSEMBLER FORMAT STR D, 'REG T', 'ADR'NL FORMATDESCRIPTION

The contents of 'REG T' are stored in
 the memory location given by 'ADDRESS'.
 The 'S' field is not used.

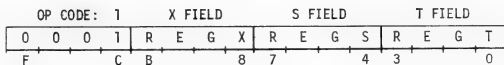
$$(\text{ADDRESS}) \leftarrow (\text{REG T})$$

- NOTE: 1. Due to the pipelined architecture of the ATAC-16M, the stored instruction should not be used to modify the instruction immediately following it.
2. This is the same as instruction STRM D, 'ADR', 1. Hence, the S-field is a count (not a "don't care").

STORE DIRECT-INDEXEDMODE DESIGNATOR DXASSEMBLER FORMAT STR DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The contents of 'REG T' are stored in the address formed by adding 'BASE' and 'REG S'.

$$(BASE + (REG S)) \leftarrow (REG T)$$

STORE REGISTER INDEXEDMODE DESIGNATOR RXASSEMBLER FORMAT STR RX, 'REG T', 'REG S', 'REG X'RX FORMATDESCRIPTION

The contents of 'REG T' are stored in the address formed by adding the contents of 'REG X' and 'REG S'.

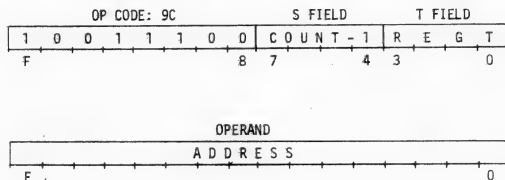
$$((\text{REG X}) + (\text{REG S})) + (\text{REG T})$$

STORE MULTIPLE

MODES: D, DR

CONDITION CODE: The condition code is not changed
by execution of these instructions.

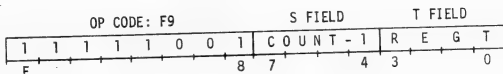
STRM

STORE MULTIPLEMODE DESIGNATOR DASSEMBLER FORMAT STRM D, 'REG T', 'ADR', 'COUNT'NL FORMATDESCRIPTION

The contents of 'COUNT' consecutive registers beginning with 'REG T' are stored into the consecutive memory locations beginning with 'ADDRESS'. 'COUNT' is the number of registers to be stored. The condition code is not changed by this instruction.

(REG T) THRU (REG T + 'COUNT' - 1)* STORED IN MEMORY LOCATIONS:
'ADDRESS' THRU 'ADDRESS' + 'COUNT' - 1.

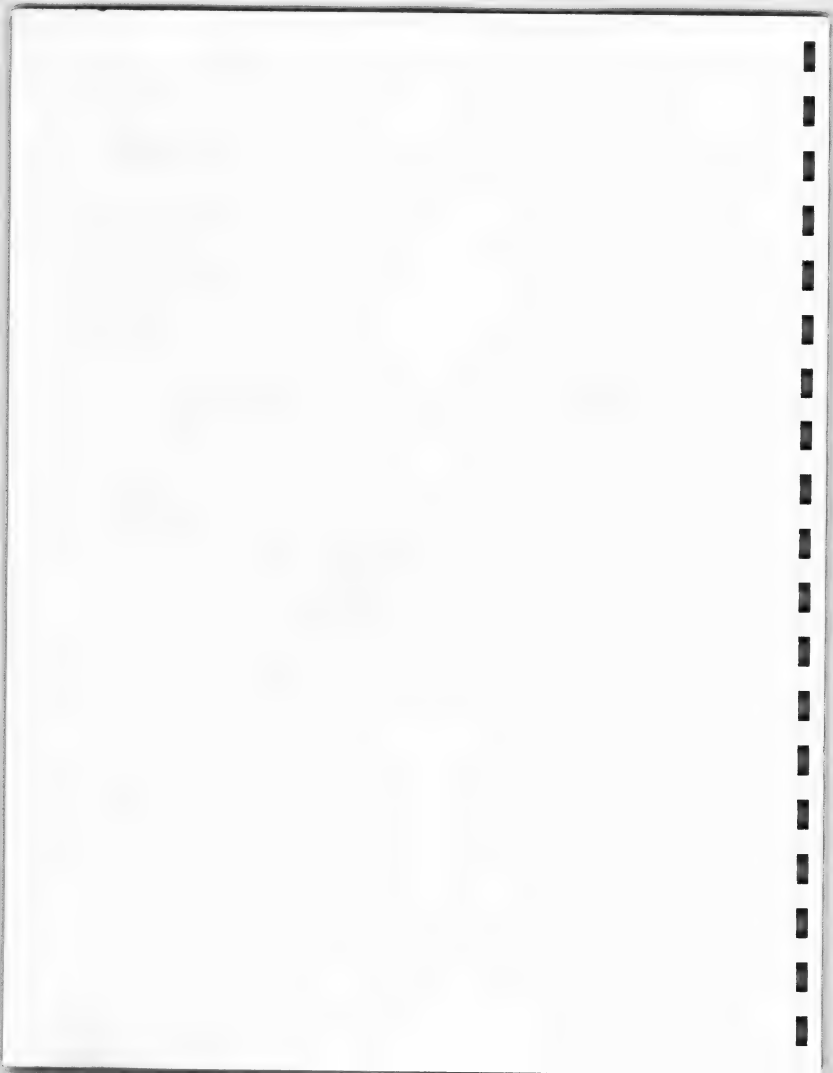
*MODULO 16

STORE MULTIPLEMODE DESIGNATOR DRASSEMBLER FORMAT STRM DR, 'REG T', 'COUNT'RR FORMATDESCRIPTION

The contents of 'COUNT' consecutive registers beginning with 'REG T' + 1 are stored into the consecutive memory locations beginning with (REG T) and ending with (REG T) + 'COUNT' - 1*. 'COUNT' is the number of locations to be stored. The condition code is not changed by this instruction.

(REG T + 1) THRU (REG T + 'COUNT')* STORED IN MEMORY LOCATIONS:
 (REG T) THRU (REG T) + 'COUNT' - 1.

*MODULO 16



SUBTRACTION

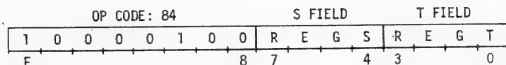
MODES: R, I, D, DX, RX, DR.

CONDITION CODE: The condition code for all subtract instructions is set by the final contents of 'REG T' as follows:

	POS	ZERO	NEG
No Overflow	0001	0010	0100
Overflow	1100	----	1001

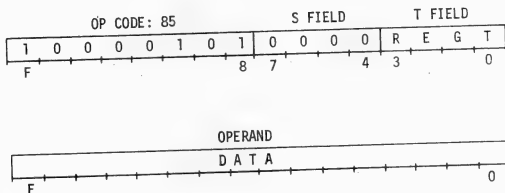
The case 8000-8000 does not produce a CCR=1010 even though the 2's complement of 8000 is 8000. This is because the subtraction is carried out as 8000 + 7FFF + 1, which produces a zero without overflow.

SUB

SUBTRACTIONMODE DESIGNATOR RASSEMBLER FORMAT SUB R, 'REG T', 'REG S'RR FORMATDESCRIPTION

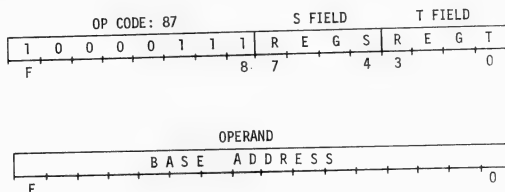
The value (subtrahend) in 'REG S' is subtracted from the value (minuend) in 'REG T'. The difference appears in 'REG T', and the condition code is set.

$\{ \text{REG T} \} \leftarrow \{ \text{REG T} \} - \{ \text{REG S} \};$
Set CCR

SUBTRACTIONMODE DESIGNATOR IASSEMBLER FORMAT SUB I, 'REG T', 'DATA'NL FORMATDESCRIPTION

The value entered as 'DATA' (subtrahend) is subtracted from the value in 'REG T' (minuend). The difference is stored in 'REG T', and the condition code is set. The 'S' field is not used.

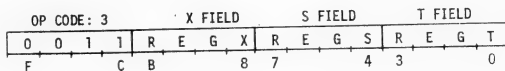
(REG T) ← (REG T) - 'DATA':
Set CCR

SUBTRACTIONMODE DESIGNATOR DXASSEMBLER FORMAT SUB DX, 'REG T', 'BASE', 'REG S'NL FORMATDESCRIPTION

The value (subtrahend) at the address formed by adding 'BASE' and 'REG S' is subtracted from the value in 'REG T' (minuend). The difference appears in 'REG T' and the condition code is set.

$$(\text{REG T}) \leftarrow (\text{REG T}) - (\text{BASE} + (\text{REG S}));$$

Set CCR

SUBTRACTIONMODE DESIGNATOR RXASSEMBLER FORMAT SUB RX, 'REG T', 'REG S', 'REG X'RX FORMATDESCRIPTION

The value (subtrahend) at the address formed by adding 'REG X' to 'REG S' is subtracted from the value in 'REG T' (minuend). The difference appears in 'REG T', and the condition code is set.

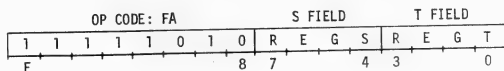
$(REG\ T) \leftarrow (REG\ T) - ((REG\ X) + (REG\ S))$
Set CCR

SUBTRACTIONMODE DESIGNATOR

DR

ASSEMBLER FORMAT

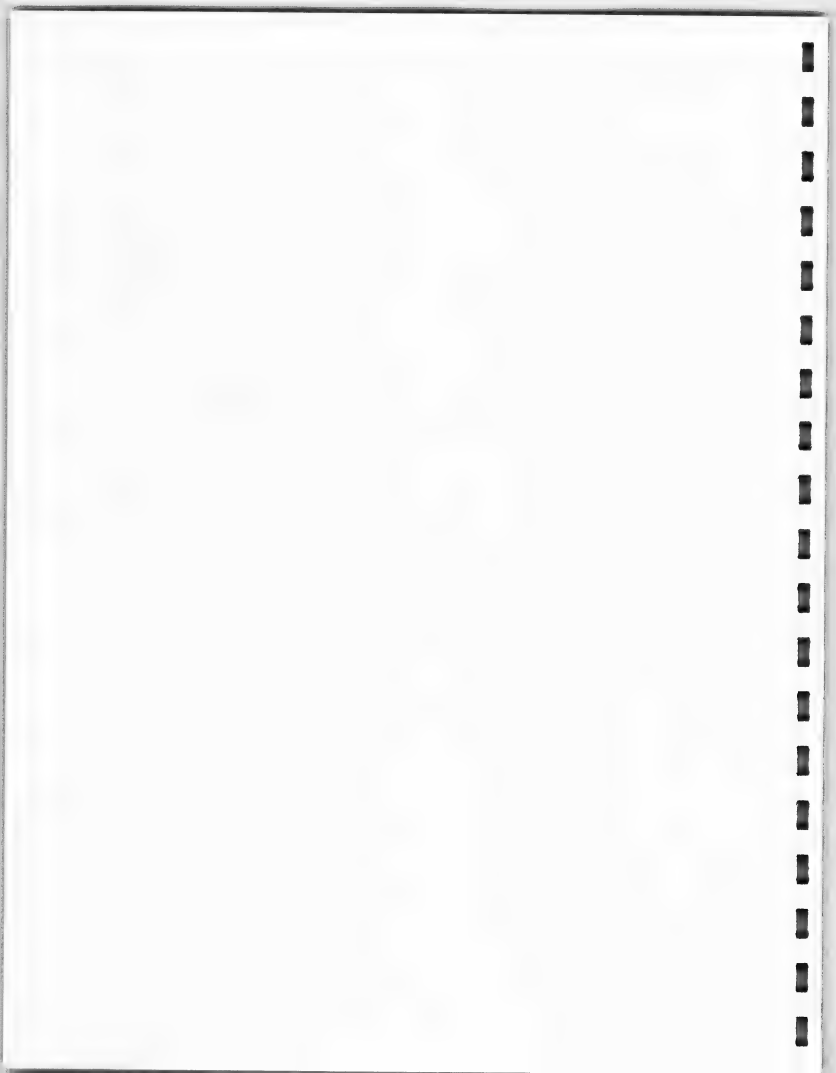
SUB DR, 'REG T', 'REG S'

RR FORMATDESCRIPTION

The value (subtrahend) at the address in 'REG S' is subtracted from the value in 'REG T' (minuend). The difference appears in 'REG T', and the condition code is set.

$$(\text{REG T}) \leftarrow (\text{REG T}) - ((\text{REG S})):$$

Set CCR



SWAP INTERRUPT MASK

MODES: R

CONDITION CODE: The condition code is not changed
by execution of the 'SWIM' instruction.

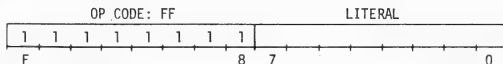
SWIM

TRAP

MODES: NONE

CONDITION CODE: The condition code is not changed
by execution of a 'TRAP' instruction.

TRAP

TRAPMODE DESIGNATOR NoneASSEMBLER FORMAT TRAP 'LIT'TRAP FORMATDESCRIPTION

Execution of a trap instruction causes the standard interrupt microcode to be executed as an instruction. Normal interrupt procedures are followed: a pointer to a three-word status save area is read from memory location 0003, the program status word is written to ((0003)), and the address of the instruction following the TRAP instruction is written to ((0003) + 1). Execution of the TRAP interrupt service routine begins at the address stored in memory location (0003) + 2. The TRAP instruction does not interface with the interrupt hardware (except for reading out the interrupt priority register); consequently, the TRAP instruction cannot be inhibited by the interrupt mask or the DSI instructions and does not affect the current interrupt level. Any interrupt level may interrupt the trap instruction's service routine (subject to the IMR and IPR), and conversely, the TRAP instruction can be used in any interrupt service routine.

The literal value 00 to FF may be used as a parameter to differentiate up to 256 unique traps to an executive program. Refer also to section 3.2.

$$((3)) \leftarrow (IPR) | (CCR)$$

$$((3) + 1) \leftarrow (PCR)$$

$$PCR \leftarrow ((3) + 2)$$

EXCHANGE REGISTERS

MODES: R

CONDITION CODE: The condition code for 'EXCHANGE
REGISTERS' is always set by the
final contents of 'REG T' as
follows:

(REG T) > 0, CONDITION CODE IS 0001

(REG T) = 0, CONDITION CODE IS 0010

(REG T) < 0, CONDITION CODE IS 0100

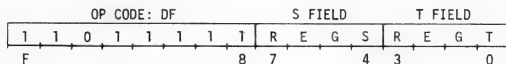
XCH

EXCHANGE REGISTERSMODE

R

ASSEMBLER FORMAT

XCH R, 'REG T', 'REG S'

RR FORMATDESCRIPTION

The contents of 'REG T' are loaded into
 'REG S', and the original contents of
 'REG S' are loaded into 'REG T'.

(REG T) ↔ (REG S):

Set CCR

EXCHANGE BYTES

MODES: R

CONDITION CODE: The condition code for the 'EXCHANGE BYTES' instruction is always set by the final contents of 'REG T' as follows:

(REG T) > 0, CONDITION CODE IS 0001

(REG T) = 0, CONDITION CODE IS 0010

(REG T) < 0, CONDITION CODE IS 0100

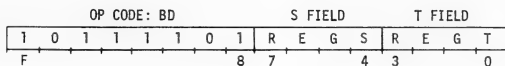
XCHB

EXCHANGE BYTESMODE

R

ASSEMBLER FORMAT

XCHB R, 'REG T', 'REG S'

RR FORMATDESCRIPTION

The high order byte (BITS F-8) of the contents of 'REG S' is loaded into the low order byte (BITS 7-0) of 'REG T'. The low order byte of 'REG S' is loaded into the high order byte of 'REG T'. The condition code is set.

(REG T, 7-0) ← (REG S, F-8);

(REG T, F-8) ← (REG S, 7-0):

Set CCR

EXCHANGE MULTIPLE

MODES: D

CONDITION CODE: The condition code is not changed
by execution of this instruction.

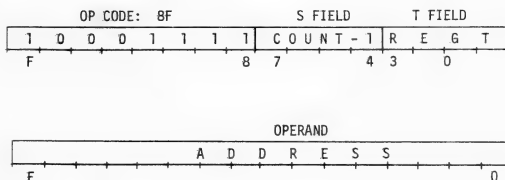
XCHM

EXCHANGE MULTIPLE DIRECTMODE

D

ASSEMBLER FORMAT

XCHM D, 'REG T', 'ADR', 'COUNT'

NL FORMATDESCRIPTION

This instruction exchanges register contents with memory locations. 'ADDRESS' defines the location in memory where the register-memory swap is to begin. 'REG T' defines the beginning register for the swap. The 'COUNT' is the number of locations and registers to be swapped.

(REG T) THRU (REG T+'COUNT'-1)* ↔

(ADDRESS) THRU (ADDRESS+'COUNT'-1)

*Modulo 16

EXECUTE MODIFIED INSTRUCTION

MODES: R

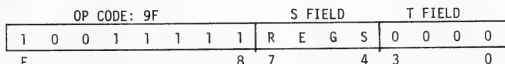
CONDITION CODE: The condition code is not changed by this instruction, but it may be changed by the execution of the modified instruction.

EXECUTE MODIFIED INSTRUCTION REGISTERMODE

R

ASSEMBLER FORMAT

XMDI R, 'REG S'

RR FORMATDESCRIPTION

The low order byte content of 'REG S' replaces the low order byte content ['REG S and 'REG T' designator values] of the next [immediate] instruction during execution of the immediate instruction.

(PC+1): REG S || REG T + (REG S₇₋₀)

EXAMPLE

XMDI	R,4	{ Assume }
SHS	LL,7,3	{ (R4) = 0068 }

This instruction sequence effectively executes as:

SHS LL,8,7

which provides a mechanism for passing 'REG S', 'REG T' values as parameters.

NOTE: Interrupts are inhibited between the XMDI and the next instruction. The XMDI instruction must not be used in conjunction with the SEARCH instructions. XMDI does not modify instructions in program storage thus protecting program re-enterability. In addition, the XMDI instruction should not be single stepped while the system is under mini-panel control, as the following instruction will be faithfully executed (i.e., will not be modified by the XMDI instruction).

EXCLUSIVE-OR

MODES: R, I, D, DX

CONDITION CODE: The condition code for an exclusive-OR operation is set based upon the results appearing in 'REG T'.

(REG T) > 0, CONDITION CODE IS 001

(REG T) = 0, CONDITION CODE IS 0010

(REG T) < 0, CONDITION CODE IS 0100

Overflow is always zero

EXCLUSIVE-OR TRUTH TABLE

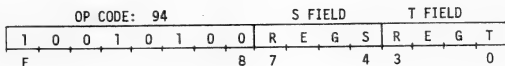
BIT S	BIT T	RESULT
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE-OR REGISTERMODE

R

ASSEMBLER FORMAT

XOR R, 'REG T', 'REG S'

RR FORMATDESCRIPTION

The contents of 'REG S' are exclusive-OR'ed with the contents of 'REG T'. The results appear in 'REG T', and the condition code is set.

$$(\text{REG T}) \leftarrow (\text{REG T}) \text{ <XOR> } (\text{REG S})$$

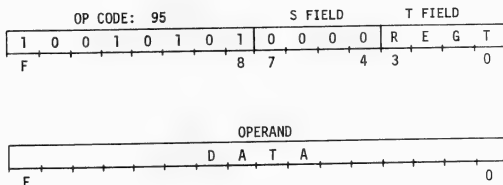
Set CCR

EXCLUSIVE-OR IMMEDIATEMODE

I

ASSEMBLER FORMAT

XOR I, 'REG T', 'DATA'

NL FORMATDESCRIPTION

The value entered as 'DATA' is exclusive-OR'ed with the contents of 'REG T'. The results appear in 'REG T', and the condition code is set. The 'S' field is not used.

(REG T) ← (REG T) <XOR> DATA

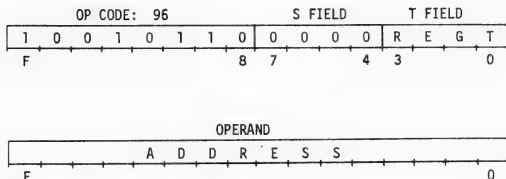
Set CCR

EXCLUSIVE-OR DIRECTMODE

D

ASSEMBLER FORMAT

XOR D, 'REG T', 'ADR'

NL FORMATDESCRIPTION

The value entered as 'ADDRESS' is treated as an address, and the contents of that address are exclusive-ORed with the contents of 'REG T'. The results appear in 'REG T', and the condition code is set. The 'S' field is not used.

$$(\text{REG T}) \leftarrow (\text{REG T}) \text{ <XOR> } (\text{ADDRESS})$$

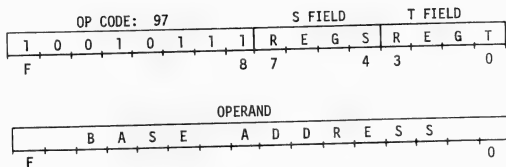
Set CCR

EXCLUSIVE-OR DIRECT INDEXEDMODE

DX

ASSEMBLER FORMAT

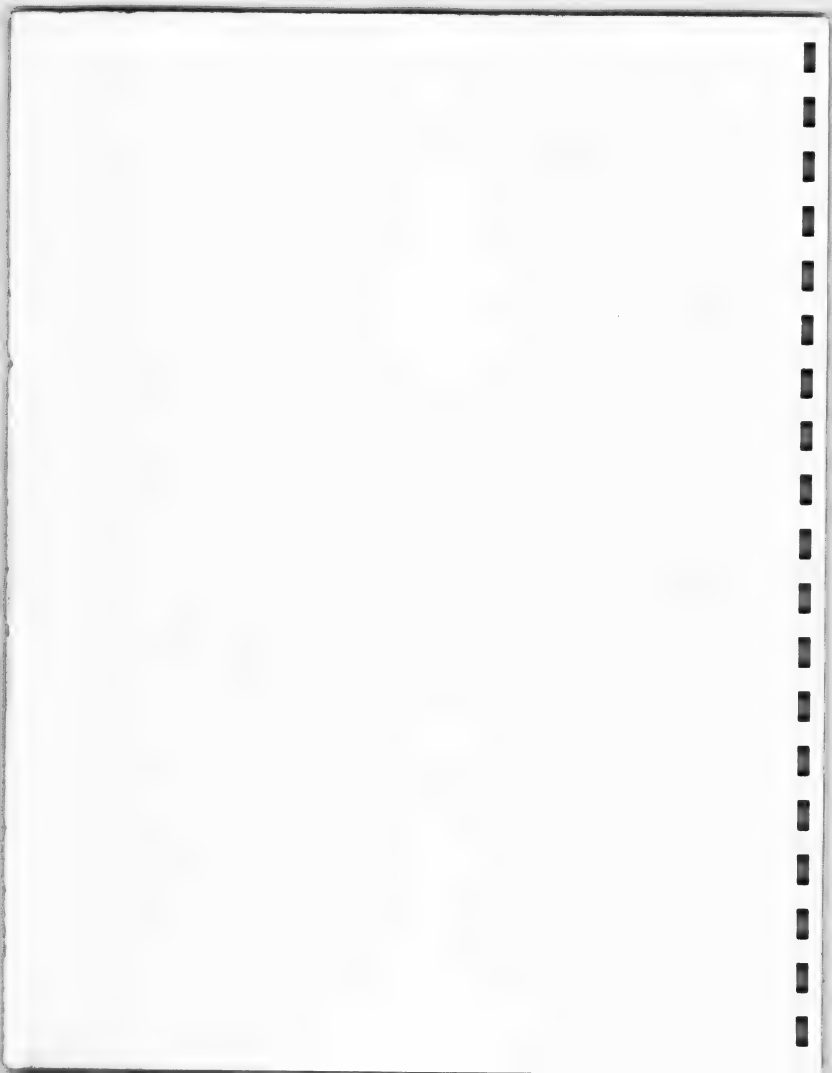
XOR DX, 'REG T', 'BASE', 'REG S'

NL FORMATDESCRIPTION

The value entered as 'BASE' is treated as a base address and the content of 'REG S' is an index taken with respect to that address. The content of the address formed by adding 'REG S' to 'BASE' is exclusive-ORed with the contents of 'REG T'. The results appear in 'REG T', and the condition code is set.

$$(\text{REG T}) \leftarrow (\text{REG T}) \text{ <XOR> } (\text{BASE} + (\text{REG S}))$$

Set CCR



APPENDIX A
SYMBOLIC CONDITION BRANCH ENTRIES

BRANCH INSTRUCTIONS WITH SYMBOLIC CONDITION ENTRIES

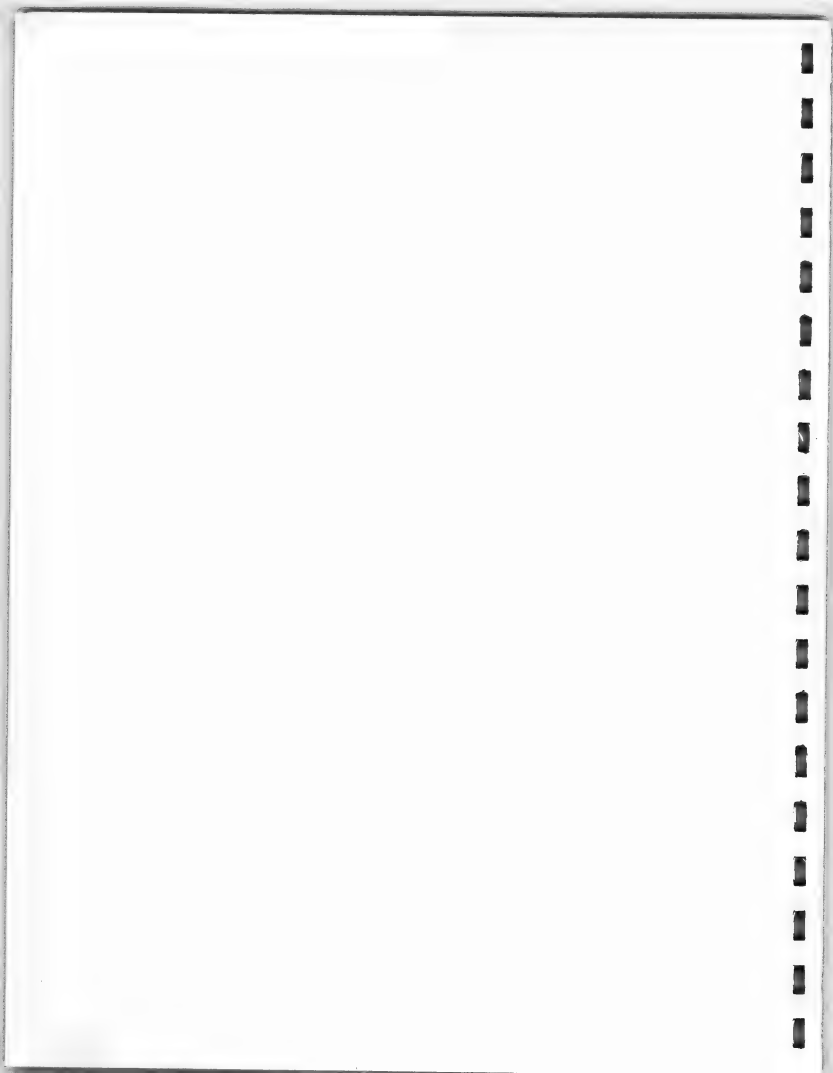
The ATAC assembler has the added capability to assemble Branch on Condition instructions which specify the branch condition mask by using a mnemonic. Typical assembler formats for BRC instructions are shown in the following list:

<u>MODE</u>	<u>ASSEMBLER FORMAT</u>	
Register	BRC	R, 'BC', 'ADRS'
Immediate	BRC	I, 'BC', 'ADRS'
or	BRCL	'BC', 'ADRS'
Direct	BRC	D, 'BC', 'ADRS'
Direct-Indexed	BRC	DX, 'BC', 'ADRS'
Short	BRC	IS, 'BC', 'ADRS'
or	BRC	PC, 'BC', 'ADRS'
or	BRCS	'BC', 'ADRS'

'BC' is any branch condition mnemonic found in the subsequent table, and 'ADRS' is any valid address expression.

Table of 'BC' (Branch Condition) Mnemonics

<u>'BC'</u> <u>Mnemonic</u>	<u>Description</u>	<u>Condition</u> <u>Mask</u> <u>(Hex)</u>
U	Unconditional jump	7
	<u>Arithmetic</u>	
P	Positive	1
N	Negative	4
NP	Not Positive	6
NN	Not Negative	3
NZ	Not Zero	5
Z	Zero	2
OV	Overflow	8
	<u>Logical</u>	
BZ	All bits '0'	2
MP	Sign bit positive	1
MN	Sign bit negative	4
	<u>Comparison</u>	
EQ	Equal	2
NE	Not Equal	5
LT	Less Than	4
GT	Greater Than	1
LE	Less Than or Equal	6
GE	Greater Than or Equal	3



APPENDIX B
INSTRUCTION SET SUMMARY

ATAC ASSEMBLER INSTRUCTIONS

<u>Instruction</u>	<u>Modes</u>	<u>Function</u>
ADD	R,I,D,DX,IS,RX,DR	Addition
AND	R,I,D,DX	Logical Product
BAL	R,I,D,DX	Branch and Link
BRC	R,I,D,DX,PC	Branch on Condition
CBL	R,D,DX	Compare Between Limits
CINT	R	Clear Interrupts
CME	R,I,D,DX	Compare Masked Equality
CMPA	R,I,D,DX	Compare Address
CMPL	R,I	Compare Logical
CMPS	R,I,D,DX,IS	Compare Signed
CMS	D,DX	Clear Memory Semaphore
DADD	R,D	Double Precision Addition
DIV	R	Division
DSI		Disable Interrupts
DSUB	R,D	Double Precision Subtraction
ENI		Enable Interrupts
FADD	R	Floating Point Addition
FDIV	R	Floating Point Division
FIX	I	Convert to Fixed Point
FLT	I	Convert to Floating Point
FMUL	R	Floating Point Multiplication
FSUB	R	Floating Point Subtraction
HALT		Halt
IBN	R,I,D,DX	Increment and Branch Negative
IOR	R,I,D,DX	Inclusive OR
LAC	R,I,D,DX	Load Arithmetic (2's) Complement
LDLB	R,I,D,DX	Load Lower Byte
LDR	R,I,D,DX,IS,RX	Load Register
LDRM	D,DR	Load Multiple
LDST	D,DX	Load Status
LDUB	R,I,D,DX	Load Upper Byte
LLC	R,I,D,DX	Load Logical (1's) Complement
MUL	R,I,D,DX	Multiplication
NOP		No Operation
RET	DR	Return Multiple Register
RIN		Register Input
ROUT		Register Output
SCHL	AL,BL	Search Lower Byte
SCHW	AL,BL	Search Word
SHD	LL,RA,RL	Shift Double Register
SHS	LL,RA,RL	Shift Single Register
SIM		ATAC Simulator Control
SMS	D,DX	Set Memory Semaphore
STK	DR	Stack Multiple Register
STR	D,DX,RX	Store

ATAC ASSEMBLER INSTRUCTIONS (continued)

<u>Instruction</u>	<u>Modes</u>	<u>Function</u>
STRM	D,DR	Store Multiple
SUB	R,I,D,DX,RX,DR	Subtraction
SWIM	R	Swap Interrupt Mask
TRAP		Initiate TRAP Interrupt
XCH	R	Exchange Registers
XCHB	R	Exchange Bytes, Register
XCHM	D	Exchange Multiple
XMDI	R	Exclusive OR
XOR	R,I,D,DX	Execute Modified Instruction
		Exclusive OR

ATAC ASSEMBLER INSTRUCTIONS: Grouped by Operation

<u>Type</u>	<u>Name</u>	<u>Modes</u>	<u>Function</u>
ARITHMETIC: Single Precision, Fixed Point			
	ADD	R,I,D,DX,IS,RX,DR	Addition
	SUB	R,I,D,DX,RX,DR	Subtraction
	MUL	R,I,D,DX	Multiplication
	DIV	R	Division
Double Precision, Fixed Point			
	DADD	R,D	Addition
	DSUB	R,D	Subtraction
Floating Point			
	FADD	R	Addition
	FSUB	R	Subtraction
	FMUL	R	Multiplication
	FDIV	R	Division
	FLT	I	Convert to Floating Point
	FIX	I	Convert to Fixed Point
LOGICAL	AND	R,I,D,DX	Logical Product
	IOR	R,I,D,DX	Inclusive OR
	XOR	R,I,D,DX	Exclusive OR
COMPARE	CMPS	R,I,D,DX,IS	Compare Signed
	CMPA	R,I,D,DX	Compare Address (Unsigned)
	CBL	R,D,DX	Compare Between Limits
	CME	R,I,D,DX	Compare Masked Equality
	CMPL	R,I	Compare Logical
SEARCH	SCHL	AL,BL	Search Lower Byte
	SCHW	AL,BL	Search Word
DATA TRANSFER: Single Words			
	LDR	R,I,D,DX,IS,RX	Load Register
	LAC	R,I,D,DX	Load Arithmetic (2's) Complement
	LLC	R,I,D,DX	Load Logical (1's) Complement
	STR	D,DX,RX	Store Register
	XCH	R	Exchange Registers
DATA TRANSFER: Bytes			
	LDLB	R,I,D,DX	Load Lower Byte
	LDUB	R,I,D,DX	Load Upper Byte
	XCHB	R	Exchange Bytes

ATAC ASSEMBLER INSTRUCTIONS: Grouped by Operation (Continued)

<u>Type</u>	<u>Name</u>	<u>Modes</u>	<u>Function</u>
DATA TRANSFER: Multiple Words / Non-Stack			
	LDRM	D,DR	Load Multiple
	STRM	D,DR	Store Multiple
	XCHM	D	Exchange Multiple
DATA TRANSFER: Multiple Words / Stack			
	STK	DR	Stack Multiple Registers
	RET	DR	Return Multiple Registers
BRANCH	BRC	R,I,D,DX,PC	Branch on Condition
	BAL	R,I,D,DX	Branch and Link
	IBN	R,I,D,DX	Increment and Branch Negative
SHIFT	SHS	LL,RA,RL	Shift Single Register
	SHD	LL,RA,RL	Shift Double Register
I/O	RIN		Register Input
	ROUT		Register Output
INTERRUPT	ENI		Enable Interrupts
	DSI		Disable Interrupts
	CINT	R	Clear Interrupts
	SWIM	R	Swap Interrupt Mask
	TRAP		Initiate TRAP Interrupt
	LDST	D,DX	Load Status
SEMAPHORE	SMS	D,DX	Set Memory Semaphore
	CMS	D,DX	Clear Memory Semaphore
MISCELLANEOUS			
	XMDI	R	Execute Modified Instruction
	HALT		Halt ATAC
	NOP		No Operation
	SIM		ATAC Simulator Control

ATAC-16M INSTRUCTION OP CODE MAP, 136 instructions.
- 7 spares

MAPPING FROM:

129 used

MOST SIGNIFICANT 4 BITS OF OP CODE

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	BRC RC	STH HX	CMPS IS	SUB RK	LDR IS	LDR RX	ADD IS	ADD RX	ADUL R	LAC R	AND R	CMPA R	SMS D	CME R	LDR R	HALT	0
1									ADUL I	LAC I	AND I	CMPA I	SMS I	CME I	LDR I	CMC DX	1
2									ADD D	LAC D	AND D	CMPA D	SMS D	CME D	LDR D	FMUL R	2
3									ADD DX	LAC DX	AND DX	CMPA DX	SMS DX	CME DX	LDR DX	FDIV R	3
4									SUB R	XOR R	IOR R	LLC R	LDLB R	LDUB R	FADD R	CMPS R	4
5									SUB I	XOR I	IOR I	LLC I	LDLB I	LDUB I	FSUB R	CMPS I	5
6									SUB D	XOR D	IOR D	LLC D	LDLB D	LDUB D	FLT I	CMPS D	6
7									SUB DX	XOR DX	IOR DX	LLC DX	LDLB DX	LDUB DX	FIX I	CMPS DX	7
8									CBL R	LDST DX	DSUB D	SCHL BL	IBN R	SHD RA	SHD RL	SHD LL	8
9									CINT I	LESL D	LDADD I	SCHW BI	IBN I	HORIT I	(Spares) ✓	SEHM DH	9
A									CBL I	SWIM R	DSUB R	CMS D	IBN D	CMS DX	(Spares) ✓	SUB DR	A
B									CBL DX	STR DX	DADD R	SCHW AL	IBN DX	SCHL AL	(Spares) ✓	LDIM DR	B
C									RIN	STR DM	SHS RA	LDIM D	MUL R	(Spares) ✓	BAL R	DIV R	C
D									STK DR	CMPL R	SHS RL	XCHB R	MUL I	(Spares) ✓	BAL I	(Spares) ✓	D
E									RET DR	CMPL I	SHS LL	ADD DR	MUL D	(Spares) ✓	BAL D	SIM (NOP)	E
F									XCHM D	XMDI R	DSI ENI	BRC R	MUL DX	XCH R	BAL DX	TRAP	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

7611 4949 401

ATAC-16M Instruction Op Code Map

APPENDIX C
ATAC-16M INSTRUCTION EXECUTION TIMES

This appendix presents instruction timing examples for the three different memory configurations shown in the table starting on page C-2. The PMU-16, RAM-16 and MRU-8 memories are standard ATAC products. Further information on these memories may be obtained by requesting the relevant publications:

<u>Product</u>	<u>Publication Number</u>
PMU-16	DS-242
RAM-16	52-056127
MRU-8	DS-243

The core memory parameters used here represent those available in a typical, full-military-temperature-range core memory.

$$\begin{array}{r} 16 \\ 16 \\ \hline 32 \end{array}$$

/
 /
 / TIMING INFORMATION FOR THREE TYPICAL MEMORY
 / CONFIGURATIONS IS SHOWN IN THE THREE COLUMNS BELOW.
 /

/ C1: INSTRUCTION MEMORY = PMU-16
 / DATA MEMORY = RAM-16
 /

/ C2: INSTRUCTION MEMORY = PMU-16
 / DATA MEMORY = MRU-8 -- ISOLATED ACCESS
 / W/O CONTENTION
 /

/ C3: INSTRUCTION MEMORY = CORE -- FULL
 / DATA MEMORY = CORE MILITARY
 / TEMPERATURE RANGE
 /

/ THE TERMS 'KIRC', 'KIWC' AND 'MAK' USED BELOW ARE
 / DEFINED IN THE MEMORY INTERFACE FIGURES 1-1 AND 1-2.
 /

	C1	C2	C3	
	----	----	----	
/ CPU :				
BASIC CYCLE TIME	BCT = 250	250	250 NS	
KIRC DELAY TIME	KIRCDL = 7	7	7 NS	(CYCLE START TO KIRC)
KIWC DELAY TIME	KIWCDL = 70	70	70 NS	(CYCLE START TO KIWC)
/ INSTRUCTION MEMORY :				
READ REQUEST DELAY	IMRDLY = 0	0	63 NS	(CYCLE START TO KIRC)
READ ACCESS TIME	IMRACC = 125	125	350 NS	(KIRC TO MAK)
READ CYCLE TIME	IMRCYC = 125	125	900 NS	(KIRC TO MEM.FREE)
/ DATA MEMORY :				
READ REQUEST DELAY	DMRDLY = 0	0	63 NS	(CYCLE START TO KIRC)
READ ACCESS TIME	DMRACC = 125	470	350 NS	(KIRC TO MAK)
READ CYCLE TIME	DMRCYC = 125	470	900 NS	(KIRC TO MEM.FREE)
WRITE REQUEST DELAY	DMWDLY = 0	0	63 NS	(CYCLE START TO KIWC)
WRITE RELEASE TIME	DMWRLS = 75	470	350 NS	(KIWC TO MAK)
WRITE CYCLE TIME	DMWCYC = 75	470	900 NS	(KIWC TO MEM.FREE)

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
/          C1: PMU-16  RAM-16  C2: PMU-16  MRU-8   C3: COPE   CORE
/

```

ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM

```

INSTR  D E S C R I P T I O N  C C M D O P C D  E X E C U T I O N T I M E I N U S E C
-----R-----C1-----C2-----C3-----
//////////

```

O P C O D E N O T E S

```

/      M = CCR MASK
/      S = SOURCE REGISTER
/      T = DESTINATION REGISTER
/      R = RX MODE REGISTER
/      N/NN = ANY HEX NUMBER
/      LL = LITERAL
/           (SIGN EXTENDED BYTE)
/      O-F = HEX NUMBER SHOWN
/      X = DON'T CARE
/           (ASSEMBLES AS 0)
/

```

```

/      ALL I,D & DX MODES ARE 2-WORD
/      INSTRUCTIONS
/

```

```

/      A MODE DESIGNATION OF '---'
/      INDICATES 'NO MODE'
/

```

```

/      '*' BEFORE THE MODE DESIGNATION
/      MEANS THE CCR IS SET
/

```

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
/      C1: PMU-16  RAM-16  C2: PMU-16  MRU-8   C3: CORE   CORE
/
/      ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM
/
INSTR  D E S C R I P T I O N  CC MD DPCD  EXECUTION TIME IN USEC
-----R-----C1-----C2-----C3--
//////////

```

```

/  A R I T H M E T I C
/  SINGLE PRECISION, FIXED POINT
/

```

INSTR	DESCRIPTION	CC	MD	DPCD	C1	C2	C3
ADD	ADDITION	* R	80ST		0.250	0.250	0.900
ADD	ADDITION	* I	81XT		0.500	0.500	1.800
ADD	ADDITION	* D	82XT		0.750	1.063	2.700
ADD	ADDITION	* DX	83ST		0.750	1.063	2.700
ADD	ADDITION	* IS	6LLT		0.250	0.250	0.900
ADD	ADDITION	* RX	7RST		0.750	1.063	2.050
ADD	ADDITION	* DR	8EST		0.750	1.063	2.050

SUB	SUBTRACTION	* R	84ST		0.250	0.250	0.900
SUB	SUBTRACTION	* I	85XT		0.500	0.500	1.800
SUB	SUBTRACTION	* D	86XT		0.750	1.063	2.700
SUB	SUBTRACTION	* DX	87ST		0.750	1.063	2.700
SUB	SUBTRACTION	* RX	3RST		0.750	1.063	2.050
SUB	SUBTRACTION	* DR	FAST		0.750	1.063	2.050

MUL	MULTIPLICATION	* R	CCST		5.500	5.500	5.500
MUL	MULTIPLICATION	* I	CDXT		5.750	5.750	6.400
MUL	MULTIPLICATION	* D	CEXT		6.000	6.313	7.300
MUL	MULTIPLICATION	* DX	CFST		6.000	6.313	7.300

DIV	DIVISION	* R	FCST		11.250	11.250	11.250
-----	----------	-----	------	--	--------	--------	--------

```

/  A R I T H M E T I C
/  DOUBLE PRECISION, FIXED POINT
/

```

DADD	DOUBLE PRECISION ADD	* R	ABST		0.750	0.750	1.400
DADD	DOUBLE PRECISION ADD	* D	A9XT		1.250	1.875	3.850
DSUB	DOUBLE PRECISION SUBTRACT	* R	AAST		0.750	0.750	1.400
DSUB	DOUBLE PRECISION SUBTRACT	* D	ABXT		1.250	1.875	3.850

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
C1: PMU-16 RAM-16 C2: PMU-16 MRU-8 C3: CORE CORE
/
/
/ ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM
/
INSTR DESCRIPTION CC MD DPCD EXECUTION TIME IN USEC
/ P --- C1 --- C2 --- C3 ---
//////////
/
/ A R I T H M E T I C
/ FLOATING POINT
/
/ A = # OF PRE-ADD ALIGN SHIFTS
/ P = # OF POST-ADD NORM SHIFTS
/ ASSUME NEITHER OPERAND = 0
/ FADD:
/ EC = NA + NP
/ NA = 13 + A : (T) >= (S)*
/ = 14 + A : (T) < (S)*
/ * EXPONENT COMPARISON
/ NP = 3 : RESULT = 0
/ = 6 : MANT OVFLOW
/ = 7 + 3P : OTHERWISE
/ FSUB: NA = 13 + A : BOTH CASES
/
/ CASE: A = 1, P = 0
FADD FLOATING POINT ADD * R E4ST 5.750 5.750 6.400
FSUB FLOATING POINT SUB * R E5ST 5.500 5.500 6.150
/
FMUL FLOATING POINT MULTIPLY * R F2ST 16.000 16.000 16.650
FDIV FLOATING POINT DIVIDE * R F3ST 28.500 28.500 29.150
/
/ N = # OF SHIFTS
EC = 3N + 8:
FLT FLOATING POINT FLOAT N=1 * I E6ST 3.250 3.250 4.550
FLT FLOATING POINT FLOAT N=2 * I E6ST 4.000 4.000 5.300
FLT FLOATING POINT FLOAT N=3 * I E6ST 4.750 4.750 6.050
...
FLT FLOATING POINT FLOAT N=30 * I E6ST 25.000 25.000 26.300
/
/ N = # OF SHIFTS
EC = 2N + 8:
FIX FLOATING POINT FIX N=1 * I E7ST 3.000 3.000 4.300
FIX FLOATING POINT FIX N=2 * I E7ST 3.500 3.500 4.800
FIX FLOATING POINT FIX N=3 * I E7ST 4.000 4.000 5.300
...
FIX FLOATING POINT FIX N=30 * I E7ST 17.500 17.500 18.800

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
 / C1: PMU-16 RAM-16 C2: PMU-16 MRU-8 C3: CORE CORE
 /

ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM

INSTR	DESCRIPTION	CC	MD	OPCD	EXECUTION TIME IN USEC		
		R			C1	C2	C3

LOGICAL GROUP

AND	LOGICAL PRODUCT	* R		A0ST	0.250	0.250	0.900
AND	LOGICAL PRODUCT	* I		A1XT	0.500	0.500	1.800
AND	LOGICAL PRODUCT	* D		A2XT	0.750	1.063	2.700
AND	LOGICAL PRODUCT	* DX		A3ST	0.750	1.063	2.700
IOR	INCLUSIVE OR	* R		A4ST	0.250	0.250	0.900
IOR	INCLUSIVE OR	* I		A5XT	0.500	0.500	1.800
IOR	INCLUSIVE OR	* D		A6XT	0.750	1.063	2.700
IOR	INCLUSIVE OR	* DX		A7ST	0.750	1.063	2.700
XOR	EXCLUSIVE OR	* R		94ST	0.250	0.250	0.900
XOR	EXCLUSIVE OR	* I		95XT	0.500	0.500	1.800
XOR	EXCLUSIVE OR	* D		96XT	0.750	1.063	2.700
XOR	EXCLUSIVE OR	* DX		97ST	0.750	1.063	2.700

COMPARE GROUP

CMPS	COMPARE SIGNED	* R		F4ST	0.250	0.250	0.900
CMPS	COMPARE SIGNED	* I		F5XT	0.500	0.500	1.800
CMPS	COMPARE SIGNED	* D		F6XT	0.750	1.063	2.700
CMPS	COMPARE SIGNED	* DX		F7ST	0.750	1.063	2.700
CMPS	COMPARE SIGNED	* IS		2LLT	0.250	0.250	0.900
CMPA	COMPARE ADDRESS (UNSIGNED)	* R		B0ST	0.750	0.750	1.400
CMPA	COMPARE ADDRESS (UNSIGNED)	* I		B1XT	1.000	1.000	2.300
CMPA	COMPARE ADDRESS (UNSIGNED)	* D		B2XT	1.250	1.563	3.200
CMPA	COMPARE ADDRESS (UNSIGNED)	* DX		B3ST	1.250	1.563	3.200

BELOW:

CBL	COMPARE BETWEEN LIMITS	* R		88ST	0.750	0.750	1.400
CBL	COMPARE BETWEEN LIMITS	* D		8AXT	1.250	1.875	3.850
CBL	COMPARE BETWEEN LIMITS	* DX		88ST	1.250	1.875	3.850

MATCH OR ABOVE:

CBL	COMPARE BETWEEN LIMITS	* R		88ST	1.250	1.250	1.900
CBL	COMPARE BETWEEN LIMITS	* D		8AXT	1.750	2.375	4.350
CBL	COMPARE BETWEEN LIMITS	* DX		88ST	1.750	2.375	4.350

CME	COMPARE MASKED EQUALITY	* R		D0ST	0.750	0.750	1.400
CME	COMPARE MASKED EQUALITY	* I		D1XT	1.000	1.000	2.300
CME	COMPARE MASKED EQUALITY	* D		D2XT	1.250	1.563	3.200
CME	COMPARE MASKED EQUALITY	* DX		D3ST	1.250	1.563	3.200

CMPL	COMPARE LOGICAL	* R		90ST	0.750	0.750	1.400
CMPL	COMPARE LOGICAL	* I		9EXT	1.000	1.000	2.300

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
/      C1: PMU-16  RAM-16  C2: PMU-16  MRU-8   C3: CORE   CORE
/
/      ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM
/
INSTR  DESCRIPTION  CC MD OPCODE  EXECUTION TIME IN USEC
-----R-----C1-----C2-----C3-----
//////////
/
/  SEARCH GROUP
/
/W = MAX # WORDS INSPECTED = 1
/W = TRUNCATED QUOTIENT
/  (IT+N-1)/N
/  WHERE N = 2-ND WORD OF INSTR
/  IT = -(REG T) INITIAL
/  DF = W      EC = 5W+4:
SCHL  SEARCH LOWER BYTE ABOVE LIM * AL DBST  2.750  3.063  4.050
SCHL  SEARCH LOWER BYTE BELOW LIM * BL BBST  2.750  3.063  4.050
/  DF = W      EC = 3W+4:
SCHW  SEARCH WORD ABOVE LIMIT * AL BBST  2.250  2.563  3.550
SCHW  SEARCH WORD BELOW LIMIT * BL B9ST  2.250  2.563  3.550
/W = MAX # WORDS INSPECTED = 32
SCHL  SEARCH LOWER BYTE ABOVE LIM * AL DBST  49.250  59.250  70.700
SCHL  SEARCH LOWER BYTE BELOW LIM * BL BBST  49.250  59.250  70.700
SCHW  SEARCH WORD ABOVE LIMIT * AL BBST  33.250  43.250  54.700
SCHW  SEARCH WORD BELOW LIMIT * BL B9ST  33.250  43.250  54.700
/W = MAX # WORDS INSPECTED = 64
SCHL  SEARCH LOWER BYTE ABOVE LIM * AL DBST  97.250  117.250  139.500
SCHL  SEARCH LOWER BYTE BELOW LIM * BL BBST  97.250  117.250  139.500
SCHW  SEARCH WORD ABOVE LIMIT * AL BBST  65.250  85.250  107.500
SCHW  SEARCH WORD BELOW LIMIT * BL B9ST  65.250  85.250  107.500

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....			
C1: PMU-16 RA4-16 C2: PMU-16 MRU-8 C3: CORE CORE			
ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM			
INSTR	DESCRIPTION	CC MD OPCODE	EXECUTION TIME IN USFC
		R ---	C1-----C2-----C3--
//////////			
DATA TRANSFER			
SINGLE WORDS			
LDR	LOAD REGISTER (S)=>(T)	* R E0ST	0.250 0.250 0.900
LDR	LOAD REGISTER 'DATA'=>(T)	* I E1XT	0.500 0.500 1.800
LDR	LOAD REGISTER (ADDR)=>(T)	* D E2XT	0.750 1.063 2.700
LDR	LOAD REG (BASE+(S))=>(T)	* DX E3ST	0.750 1.063 2.700
LDR	LOAD REGISTER 'LIT'=>(T)	* IS 4LLT	0.250 0.250 0.900
LDR	LOAD REG ((R)+(S))=>(T)	* RX 5RST	0.750 1.063 2.050
LAC	LOAD ARITHMETIC COMPLEMENT	* R 90ST	0.250 0.250 0.900
LAC	LOAD ARITHMETIC COMPLEMENT	* I 91XT	0.500 0.500 1.800
LAC	LOAD ARITHMETIC COMPLEMENT	* D 92XT	0.750 1.063 2.700
LAC	LOAD ARITHMETIC COMPLEMENT	* DX 93ST	0.750 1.063 2.700
LLC	LOAD LOGICAL COMPLEMENT	* R B4ST	0.250 0.250 0.900
LLC	LOAD LOGICAL COMPLEMENT	* I B5XT	0.500 0.500 1.800
LLC	LOAD LOGICAL COMPLEMENT	* D B6XT	0.750 1.063 2.700
LLC	LOAD LOGICAL COMPLEMENT	* DX B7ST	0.750 1.063 2.700
STR	STORE REGISTER (T)=>(ADDR)	D 9COT	1.000 1.375 2.950
STR	STORE REG (T)=>(BASE+(S))	DX 9BST	1.000 1.375 2.950
STR	STORE REG (T)=>((R)+(S))	RX 1RST	1.000 1.375 2.300
XCH	EXCHANGE REGISTERS	* R DFST	1.000 1.000 1.650
DATA TRANSFER			
BYTES			
LDLB	LOAD LOWER BYTE	* R C4ST	0.500 0.500 1.150
LDLB	LOAD LOWER BYTE	* I C5XT	0.750 0.750 2.050
LDLB	LOAD LOWER BYTE	* D C6XT	1.000 1.313 2.950
LDLB	LOAD LOWER BYTE	* DX C7ST	1.000 1.313 2.950
LDUB	LOAD UPPER BYTE	* R D4ST	0.750 0.750 1.400
LDUB	LOAD UPPER BYTE	* I D5XT	1.250 1.250 2.550
LDUB	LOAD UPPER BYTE	* D D6XT	1.500 1.813 3.450
LDUB	LOAD UPPER BYTE	* DX D7ST	1.500 1.813 3.450
XCHB	EXCHANGE BYTES, REGISTER	* R BDST	0.500 0.500 1.150


```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM....
/      C1: PMU-16  RAM-16  C2: PMU-16  MRU-8   C3: CORE   CORE
/
/      ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM
/
INSTR  D E S C R I P T I O N  C C M D O P C D  E X E C U T I O N T I M E I N U S E C
-----R-----C1-----C2-----C3-----
//////////
/
/  D A T A   T R A N S F E R
/  M U L T I P L E   W O R D S   /   N O N - S T A C K
/
/DF = N+1:
LDRM  LOAD REG MULTIPLE (1 REG)      D 8C0T  1.250  1.563  3.200
LDRM  LOAD REG MULTIPLE (2 REGS)     D 8C1T  1.500  2.125  4.100
LDRM  LOAD REG MULTIPLE (3 REGS)     D 8C2T  1.750  2.688  5.000
....  (N+1 REGS)                     D 8CNT
LDRM  LOAD REG MULTIPLE(16 REGS)     D 8CFT  5.000  10.000  16.700
/DF = N+1:
LDRM  LOAD REG MULTIPLE (1 REG)      DR FB0T  1.250  1.563  2.550
LDRM  LOAD REG MULTIPLE (2 REGS)     DR FB1T  1.500  2.125  3.450
LDRM  LOAD REG MULTIPLE (3 REGS)     DR FB2T  1.750  2.688  4.350
....  (N+1 REGS)                     DR FBNT
LDRM  LOAD REG MULTIPLE(16 REGS)     DR FBFT  1.400  1.900  3.090
/
/DW = N+1:
STRM  STORE REG MULTIPLE (1 REG)      D 9C0T  1.000  1.375  2.950
STRM  STORE REG MULTIPLE (2 REGS)     D 9C1T  1.250  2.000  3.850
STRM  STORE REG MULTIPLE (3 REGS)     D 9C2T  1.500  2.625  4.750
....  (N+1 REGS)                     D 9CNT
STRM  STORE REG MULTIPLE(16 REGS)     D 9CFT  4.750  10.750  16.450
/DW = N+1:
STRM  STORE REG MULTIPLE (1 REG)      DR F90T  1.000  1.375  2.300
STRM  STORE REG MULTIPLE (2 REGS)     DR F91T  1.250  2.000  3.200
STRM  STORE REG MULTIPLE (3 REGS)     DR F92T  1.500  2.625  4.100
....  (N+1 REGS)                     DR F9NT
STRM  STORE REG MULTIPLE(16 REGS)     DR F9FT  4.750  10.750  15.800
/
/DF = N+1  DW = N+1  EC = N+2:
XCHM  EXCHANGE MULT (1REG<=>1LOC)     D 8F0T  1.500  2.188  4.100
XCHM  EXCHANGE MULT (2REG<=>2LOC)     D 8F1T  2.250  3.625  6.150
XCHM  EXCHANGE MULT (3REG<=>3LOC)     D 8F2T  3.000  5.063  8.200
....  (N+1 REGS <=> N+1 LOCS)        D 8FNT
XCHM  EXCHANGE MULT ( 16 <=> 16 )     D 8FFT  12.750  23.750  34.850

```

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
/      C1: PMU-16  RAM-16  C2: PMU-16  MRU-8   C3: CORE   CORE
/
/      ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM
/
INSTR  D E S C R I P T I O N  CC MD OPCD  EXECUTION TIME IN USEC
-----R-----C1-----C2-----C3--
//////////
/
/  D A T A  T R A N S F E R
/  M U L T I P L E  W O R D S  /  S T A C K
/
/DW = N+1:
STK   STACK MULTIPLE REG (1 REG)   DR 8D0T   1.250   1.625   2.550
STK   STACK MULTIPLE REG (2 REGS)  DR 8D1T   1.500   2.250   3.450
STK   STACK MULTIPLE REG (3 REGS)  DR 8D2T   1.750   2.875   4.350
...   (N+1 REGS)                   DR 8DNT
STK   STACK MULTIPLE REG(16 REGS)  DR 8DFT   5.000  11.000  10.050
/
/DF = N+1:
RET   RETURN MULTIPLE REG(1 REG)   DR 8E0T   1.250   1.563   2.550
RET   RETURN MULTIPLE REG(2 REGS)  DR 8E1T   1.500   2.125   3.450
RET   RETURN MULTIPLE REG(3 REGS)  DR 8E2T   1.750   2.688   4.350
...   (N+1 REGS)                   DR 8ENT
RET   RETURN MULTIPLE REG(16 REGS)  DR 8EFT   1.400   1.900   3.090

```

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
/      C1: PMU-16  RAM-16  C2: PMU-16  MRU-8   C3: CORE   CORE
/
/      ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM
/
INSTR   D E S C R I P T I O N   CC MD OPCD   EXECUTION TIME IN USEC
----- P ----- C1-----C2-----C3--
//////////
/
/  B R A N C H   G R O U P
/
/BRANCH:
BRC  BRANCH ON CONDITION-(S)      R  BFSM    0.750  0.750  2.050
BRC  BRANCH ON CONDITION-ADDR     I  C1XM    1.000  1.000  2.950
BRC  BRANCH ON CONDITION-(ADDR)   D  C2XM    1.250  1.563  3.850
BRC  BRANCH ON COND-((S)+BASE)    DX C3SM    1.250  1.563  3.850
BRC  BRANCH ON COND-(PC)+LL+2     IS 0LLM    1.000  1.000  2.300
/NO BRANCH:
BRC  BRANCH ON CONDITION-(S)      R  BFSM    0.750  0.750  1.400
BRC  BRANCH ON CONDITION-ADDR     I  C1XM    1.000  1.000  2.300
BRC  BRANCH ON CONDITION-(ADDR)   D  C2XM    1.250  1.563  3.200
BRC  BRANCH ON COND-((S)+BASE)    DX C3SM    1.250  1.563  3.200
BRC  BRANCH ON COND-(PC)+LL+2     IS 0LLM    0.500  0.500  1.150
/
BAL  BRANCH AND LINK              R  ECST    1.000  1.000  2.300
BAL  BRANCH AND LINK              I  EDXT    1.250  1.250  3.200
BAL  BRANCH AND LINK              D  EEXT    1.500  1.813  4.100
BAL  BRANCH AND LINK              DX EFST    1.500  1.813  4.100
/
/BRANCH:
IBN  INCREMENT & BRANCH NEGATIVE * R  CBST    1.000  1.000  2.300
IBN  INCREMENT & BRANCH NEGATIVE * I  C9XT    1.250  1.250  3.200
IBN  INCREMENT & BRANCH NEGATIVE * D  CAXT    1.500  1.813  4.100
IBN  INCREMENT & BRANCH NEGATIVE * DX CBST    1.500  1.813  4.100
/NO BRANCH:
IBN  INCREMENT & BRANCH NEGATIVE * R  CBST    0.750  0.750  2.050
IBN  INCREMENT & BRANCH NEGATIVE * I  C9XT    1.250  1.250  2.550
IBN  INCREMENT & BRANCH NEGATIVE * D  CAXT    1.500  1.813  3.450
IBN  INCREMENT & BRANCH NEGATIVE * DX CBST    1.500  1.813  3.450

```

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
/      C1: PMU-16  RAM-16  C2: PMU-16  MRU-8  C3: CORE    CORE
/

```

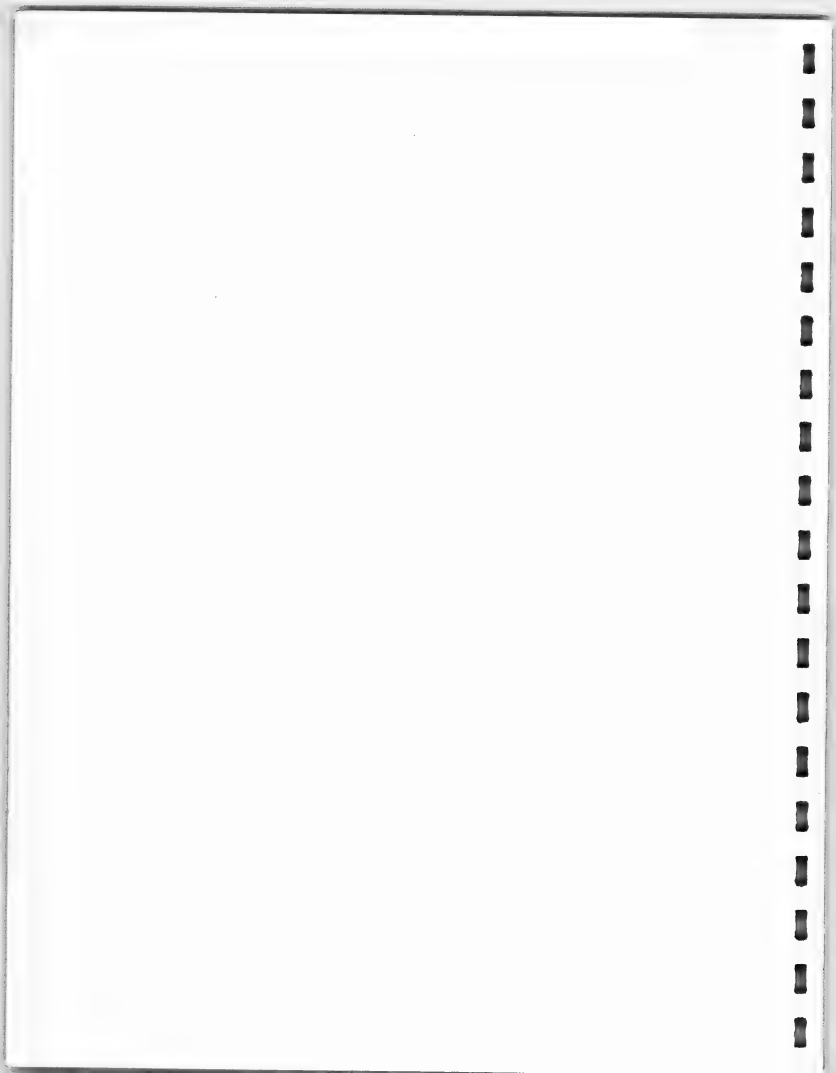
ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM

INSTR	DESCRIPTION	CC MD OPCODE	EXECUTION TIME IN USEC		
		R	C1	C2	C3
//////////					
/ SHIFT GROUP					
/					
/N=0: (SHIFT 1)					
	EC = N+2:				
SHS	SHIFT SINGLE REGISTER - LL	* LL AEOT	0.750	0.750	1.400
SHS	SHIFT SINGLE REGISTER - RL	* RL ADOT	0.750	0.750	1.400
SHS	SHIFT SINGLE REGISTER - RA	* RA ACOT	0.750	0.750	1.400
	EC = N+5:				
SHD	SHIFT DOUBLE REGISTER - LL	* LL F8OT	1.250	1.250	1.250
/					
	EC = N+6:				
SHD	SHIFT DOUBLE REGISTER - RL	* RL E8OT	1.500	1.500	1.500
SHD	SHIFT DOUBLE REGISTER - RA	* RA D8OT	1.500	1.500	1.500
/N = 1: (SHIFT 2)					
SHS	SHIFT SINGLE REGISTER - LL	* LL AE1T	1.000	1.000	1.650
SHS	SHIFT SINGLE REGISTER - RL	* RL AD1T	1.000	1.000	1.650
SHS	SHIFT SINGLE REGISTER - RA	* RA AC1T	1.000	1.000	1.650
SHD	SHIFT DOUBLE REGISTER - LL	* LL F81T	1.500	1.500	1.500
SHD	SHIFT DOUBLE REGISTER - RL	* RL E81T	1.750	1.750	1.750
SHD	SHIFT DOUBLE REGISTER - RA	* RA D81T	1.750	1.750	1.750
/N = 2: (SHIFT 3)					
SHS	SHIFT SINGLE REGISTER - LL	* LL AE2T	1.250	1.250	1.900
SHS	SHIFT SINGLE REGISTER - RL	* RL AD2T	1.250	1.250	1.900
SHS	SHIFT SINGLE REGISTER - RA	* RA AC2T	1.250	1.250	1.900
SHD	SHIFT DOUBLE REGISTER - LL	* LL F82T	1.750	1.750	1.750
SHD	SHIFT DOUBLE REGISTER - RL	* RL E82T	2.000	2.000	2.000
SHD	SHIFT DOUBLE REGISTER - RA	* RA D82T	2.000	2.000	2.000
/N=15: (SHIFT 16)					
SHS	SHIFT SINGLE REGISTER - LL	* LL AEFT	4.750	4.750	5.400
SHS	SHIFT SINGLE REGISTER - RL	* RL ADFT	4.750	4.750	5.400
SHS	SHIFT SINGLE REGISTER - RA	* RA ACFT	4.750	4.750	5.400
SHD	SHIFT DOUBLE REGISTER - LL	* LL F8FT	5.250	5.250	5.250
SHD	SHIFT DOUBLE REGISTER - RL	* RL E8FT	5.500	5.500	5.500
SHD	SHIFT DOUBLE REGISTER - RA	* RA D8FT	5.500	5.500	5.500

```

//////////.....IM.....DM...../.....IM.....DM...../.....IM.....DM.....
/      C1: PMU-16  RAM-16  C2: PMU-16  MRU-8   C3: COKE   CORE
/
/      ATAC-16M INSTRUCTION TIMING ANALYSIS PROGRAM
/
/  INSTR  DESCRIPTION  CC MD OPCODE  EXECUTION TIME IN USEC
/  -----R-----C1-----C2-----C3-----
//////////
/
/  I / O  GROUP
/
/  RIN  REGISTER INPUT      * -- 8CST    1.500    1.500    2.150
/  ROUT REGISTER OUTPUT     * -- 09ST    2.000    2.000    2.650
/
/  INTERRUPT CONTROL
/
/  ENI  ENABLE INTERRUPTS   -- AFOX    0.750    0.750    1.400
/  DIS  DISABLE INTERRUPTS  -- AFFX    0.750    0.750    1.400
/  CINT CLEAR INTERRUPTS    R  89SX    0.500    0.500    1.150
/  SWIM SWAP INTERRUPT MASK R  9AST    1.000    1.000    1.650
/
/  IENT INTERRUPT ENTRY W/O INSTR  -- ----    3.250    4.313    7.150
/      LATENCY -- THIS IS NOT AN
/      INSTRUCTION.
/  TRAP INITIATE TRAP INTERRUPT NN -- FFVN    2.750    3.813    6.650
/
/  LDST LOAD STATUS         * D  99XX    2.000    2.625    5.250
/  LDST LOAD STATUS         * DX 98SX    2.000    2.625    5.250
/
/  SEMAPHORE CONTROL
/
/  /NO CAPTURE (MIN):
/  SMS  SET MEMORY SEMAPHORE * D  COXT    1.500    1.813    3.450
/  SMS  SET MEMORY SEMAPHORE * DX F1ST    1.500    1.813    3.450
/  /NO CAPTURE (MAX):
/  SMS  SET MEMORY SEMAPHORE * D  COXT    2.250    3.188    5.500
/  SMS  SET MEMORY SEMAPHORE * DX F1ST    2.250    3.188    5.500
/  /CAPTURE:
/  SMS  SET MEMORY SEMAPHORE * D  COXT    2.250    3.250    5.500
/  SMS  SET MEMORY SEMAPHORE * DX F1ST    2.250    3.250    5.500
/
/  CMS  CLR MEMORY SEMAPHORE D  8AXT    1.250    1.938    3.850
/  CMS  CLR MEMORY SEMAPHORE DX 0AST    1.250    1.938    3.850
/
/  MISCELLANEOUS
/
/  XMDI EXECUTE MODIFIED INSTR.  R  9FSX    0.500    0.500    1.150
/  HALT HALT                    -- F030    2.000    2.000    2.650
/  NOP  NO OPERATION            -- 0X00    0.500    0.500    1.150
/  SIM  SIMULATOR CONTROL      -- FEXN    0.500    0.500    1.800
/      SIM IS 2 WORD INSTR

```



APPENDIX D
SYMBOLS, NOTATION, AND ABBREVIATIONS

SYMBOLS, NOTATION, ABBREVIATIONS

Throughout this document, certain symbols, abbreviations and notations are used to specify ATAC operations. These conventions are summarized below:

<u>Symbology</u>	<u>Definition</u>
(Item)	Contents of Item (the value of Item is an address or a register)
'Item'	Value of Item
(Item A) (Item B)	Contents of Item A concatenated with (connected to) the contents of Item B
Operator	Operators such as Logical AND are signified in brackets
+	Addition
-	Subtraction
*	Multiplication
/	Division
<AND>	Logical AND operator
<IOR>	Inclusive OR operator
<XOR>	Exclusive OR operator
<—	Value assignment
<	Less than
=	Equal to
>	Greater than

<u>Terminology</u>	<u>Definition</u>
ATAC	Applied Technology Advanced Computer
Memory Word	A 16-bit segment of ATAC program memory (bits in the word are numbered right to left, 0 to 15)
Field	A group of 1 to 16 bits within a word
Operand	A data quantity used in a computation step
Op Code	A 4 or 8 bit quantity which specified the instruction to be executed
Modulus (Modulo)	2^n , where n is the number of bits in the field of interest; $2^n - 1$ is the largest number expressable in the field
Register File	Sixteen general purpose registers, each 16 bits long
Register	One of 16 general registers in the register file; all registers can be used as index registers or accumulators
Program Memory	Up to 65,536 memory words made up of arbitrary segments of ROM, RAM, RMM, core memory types
Address	Points to one of 65,536 memory words, or 1 of 16 registers
Base Address	Address of first memory word in a sequential array of words in memory
Index Register	A register whose contents are used to form an address
Cycle	A .25 microsecond period during which a basic computation referencing two registers can be executed in ATAC-16M
Indirect Address	The address of a word in memory whose contents is another address
Program Counter Register	The register whose contents are the address of the instruction being executed
Branch	A nonsequential change in the contents of the program counter register

TerminologyDefinition

I/O Device Address

Numerical devices addresses are assigned based on individual system requirements. A 16-bit address word is available for device addressing; any level of address encoding may be built into ATAC I/O interfaces

AbbreviationsDefinitions

CP	Central processor
CCM	Condition Code Mask Value (4 bits)
CCR	Condition Code Register (4 bits)
Op Code	Instruction Operation Code
IMR	Interrupt Mask Register
MAR	Memory Address Register
MDR	Memory Data Register
PCR	Program Counter (16 bits)
RAM	Random Access (semiconductor) Memory
RegS	Source Register
RegT	Terminal (Destination) Register
RMM	Read Mostly, Electrically Alterable Memory
ROM	Read Only Memory
VS	Versus (implies comparison of two operands)

APPENDIX E
CONVERSION AIDS

Table E-1. Hexadecimal Arithmetic Tables

Addition Table

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10
2	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
3	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
4	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
5	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14
6	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15
7	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16
8	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17
9	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19
B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A
C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

MULTIPLICATION TABLE

1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
2	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
3	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
4	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
5	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
6	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
7	10	18	20	28	30	38	40	48	50	58	60	68	70	78
8	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
9	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
A	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
B	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
C	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
D	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
E	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Table E-2. Powers of Two Table

2^n	n	2^{-n}
1	0	1 0
2	1	0 5
4	2	0 25
8	3	0 125
16	4	0 062 5
32	5	0 031 25
64	6	0 015 625
128	7	0 007 812 5
256	8	0 003 906 25
512	9	0 001 953 125
1 024	10	0 000 976 562 5
2 048	11	0 000 488 281 25
4 096	12	0 000 244 140 625
8 192	13	0 000 122 070 312 5
16 384	14	0 000 061 035 156 25
32 768	15	0 000 030 517 578 125
65 536	16	0 000 015 258 789 062 5
131 072	17	0 000 007 629 394 531 25
262 144	18	0 000 003 614 897 265 625
524 288	19	0 000 001 907 348 637 812 5
1 048 576	20	0 000 000 953 674 316 406 25
2 097 152	21	0 000 000 476 837 158 203 125
4 194 304	22	0 000 000 238 418 579 101 562 5
8 388 608	23	0 000 000 119 209 289 550 781 25
16 777 216	24	0 000 000 059 604 644 775 390 625
33 554 432	25	0 000 000 029 802 322 387 695 312 5
67 108 864	26	0 000 000 014 901 161 193 847 556 25
134 217 728	27	0 000 000 007 450 580 596 923 628 125
268 435 456	28	0 000 000 003 725 290 298 461 914 062 5
536 870 912	29	0 000 000 001 862 645 149 230 557 031 25
1 073 741 824	30	0 000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0 000 000 000 465 661 287 307 739 257 312 5
4 294 967 296	32	0 000 000 000 232 830 643 653 659 628 956 25
8 589 934 592	33	0 000 000 000 116 415 321 826 934 814 531 125
17 179 869 184	34	0 000 000 000 058 207 640 913 457 407 266 562 5
34 359 738 368	35	0 000 000 000 029 103 630 456 731 703 613 281 25
68 719 476 736	36	0 000 000 000 014 551 915 228 366 851 256 640 625
137 438 953 472	37	0 000 000 000 007 275 957 614 133 425 973 320 312 5
274 877 906 944	38	0 000 000 000 003 637 978 807 631 712 531 660 156 25
549 755 813 688	39	0 000 000 000 001 818 989 403 345 896 275 630 078 125
1 099 511 627 776	40	0 000 000 000 000 909 494 701 772 928 237 915 720 662 5
2 199 023 255 552	41	0 000 000 000 000 454 747 350 805 464 118 957 519 531 25
4 398 046 511 104	42	0 000 000 000 000 227 373 675 443 232 039 478 759 765 625
8 796 093 022 208	43	0 000 000 000 000 113 686 837 721 616 019 739 379 802 812 5
17 592 186 044 416	44	0 000 000 000 000 056 843 418 659 808 074 869 899 941 406 25
35 184 372 088 032	45	0 000 000 000 000 028 421 719 613 404 017 434 841 970 703 125
70 368 744 177 664	46	0 000 000 000 000 014 210 854 715 792 033 717 422 485 351 562 5
140 737 488 355 328	47	0 000 000 000 000 007 105 427 357 001 051 658 711 242 675 781 25
281 474 976 710 656	48	0 000 000 000 000 003 552 713 670 870 520 929 355 621 337 890 625
562 949 953 421 312	49	0 000 000 000 000 001 776 356 633 450 290 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0 000 000 000 000 000 888 178 419 700 125 332 398 905 334 472 656 25
2 251 799 813 685 248	51	0 000 000 000 000 000 444 089 209 850 062 616 169 452 657 236 328 125
4 503 599 627 370 496	52	0 000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0 000 000 000 000 000 111 022 302 452 515 654 042 363 156 809 082 031 25
18 014 398 509 481 984	54	0 000 000 000 000 000 055 511 151 231 227 027 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0 000 000 000 000 000 027 755 575 615 623 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0 000 000 000 000 000 013 877 737 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0 000 000 000 000 000 006 938 938 993 217 228 377 647 657 925 567 626 953 125
288 230 376 151 711 744	58	0 000 000 000 000 000 003 469 469 465 913 513 614 188 823 828 962 783 813 476 562 5
576 460 752 303 423 488	59	0 000 000 000 000 000 001 734 734 733 425 975 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0 000 000 000 000 000 000 867 361 737 938 403 547 205 962 240 695 953 369 140 625

Table E-3. Largest Decimal Number Table

Largest Decimal Integer	Decimal Digits Req'd*	Number of Binary Bits	Largest Decimal Fraction
1	1	1	5
3	2	2	75
7	3	3	875
15	4	4	937 5
31	5	5	968 75
63	6	6	984 375
127	7	7	992 187 5
255	8	8	996 093 75
511	9	9	998 046 875
1 023	10	10	999 023 437 5
2 047	11	11	999 511 718 75
4 095	12	12	999 755 859 375
8 191	13	13	999 877 929 687 5
16 383	14	14	999 938 964 843 75
32 767	15	15	999 969 482 421 875
65 535	16	16	999 984 741 210 937 5
131 071	17	17	999 992 370 605 468 75
262 143	18	18	999 996 185 302 734 375
524 287	19	19	999 998 092 651 367 187 5
1 048 575	20	20	999 999 046 325 683 593 75
2 097 151	21	21	999 999 523 162 841 796 875
4 194 303	22	22	999 999 761 581 420 898 437 5
8 388 607	23	23	999 999 880 790 710 449 218 75
16 777 215	24	24	999 999 940 395 355 244 609 375
33 554 431	25	25	999 999 970 197 677 612 304 637 5
67 108 863	26	26	999 999 985 098 638 806 152 313 75
134 217 727	27	27	999 999 992 549 419 403 076 171 875
268 435 455	28	28	999 999 996 274 709 701 538 055 937 5
536 870 911	29	29	999 999 998 137 354 850 769 042 968 75
1 073 741 823	30	30	999 999 999 068 677 425 384 521 484 375
2 147 483 647	31	31	999 999 999 534 338 712 692 263 742 187 5
4 294 967 295	32	32	999 999 999 767 169 356 346 133 371 093 75
8 589 934 591	33	33	999 999 999 883 584 678 173 065 185 546 875
17 179 869 183	34	34	999 999 999 941 792 339 066 532 592 773 437 5
34 359 738 367	35	35	999 999 999 970 896 169 543 265 296 386 718 75
68 719 476 735	36	36	999 999 999 985 448 034 771 633 148 193 359 375
137 438 953 471	37	37	999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943	38	38	999 999 999 996 362 021 192 928 287 040 339 843 75
549 755 813 887	39	39	999 999 999 998 181 010 595 451 143 524 169 921 875
1 099 511 627 775	40	40	999 999 999 999 030 505 298 227 071 762 034 960 937 5
2 199 023 255 551	41	41	999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103	42	42	999 999 999 999 772 626 324 555 767 940 521 240 234 375
8 796 093 022 207	43	43	999 999 999 999 886 313 162 273 383 970 260 620 117 187 5
17 592 186 044 415	44	44	999 999 999 999 943 156 581 133 191 985 130 310 058 593 75
35 184 372 088 831	45	45	999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 663	46	46	999 999 999 999 985 789 145 261 797 996 282 577 514 648 437 5
140 737 488 355 327	47	47	999 999 999 999 992 894 572 612 398 998 141 288 757 324 218 75
281 474 976 710 655	48	48	999 999 999 999 996 447 206 321 199 499 070 644 378 662 109 375
562 949 953 421 311	49	49	999 999 999 999 998 223 643 161 509 749 535 322 189 311 054 687 5
1 125 899 906 842 623	50	50	999 999 999 999 999 111 821 563 299 874 767 661 094 665 527 343 75
2 251 799 813 685 247	51	51	999 999 999 999 999 555 910 793 149 937 383 830 547 332 763 671 875
4 503 599 627 370 495	52	52	999 999 999 999 999 777 955 355 074 968 691 915 273 666 381 835 937 5
9 007 199 234 740 991	53	53	999 999 999 999 999 888 977 657 537 484 345 957 636 833 130 917 968 75
18 014 398 459 481 983	54	54	999 999 999 999 999 944 468 862 768 742 172 978 818 416 595 458 984 375
36 028 797 018 963 967	55	55	999 999 999 999 999 972 244 423 384 371 086 489 409 208 297 729 492 187 5
72 057 594 037 927 935	56	56	999 999 999 999 999 986 122 212 192 185 543 244 704 604 148 864 746 093 75
144 115 188 075 855 871	57	57	999 999 999 999 999 993 061 105 096 092 771 622 352 302 074 432 373 046 875
288 230 376 151 711 743	58	58	999 999 999 999 999 996 530 553 048 046 385 811 176 151 037 216 186 523 437 5
576 460 752 303 423 487	59	59	999 999 999 999 999 998 265 275 524 023 192 905 588 075 518 608 031 261 718 75
1 152 921 504 606 846 975	60	60	999 999 999 999 999 999 132 632 262 011 596 452 794 037 759 304 046 630 859 375

*Larger numbers within a digit group should be checked for exact number of decimal digits required.

Examples of use:

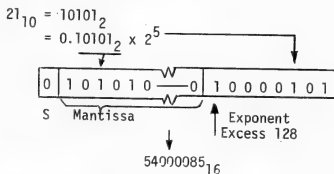
- Q. What is the largest decimal value that can be expressed by 36 binary digits?
A. 68,719,476,735
- Q. How many decimal digits will be required to express a 22-bit number?
A. 7 decimal digits.

APPENDIX F

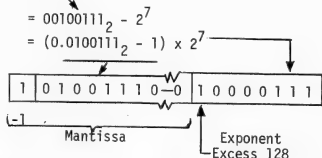
FLOATING POINT NUMBERS -
CONVERSION AND EXAMPLES

Table F-1 gives several useful examples of floating point numbers. In what follows, several of these numbers will be analyzed to illustrate the ideas discussed in section 6.1.

1. Converting to Floating Point



$$-89_{10} = 39_{10} - 128_{10} \quad (128 \text{ is first power of } 2 \geq 89)$$



2. Converting from Floating Point

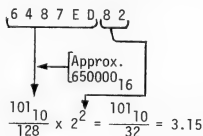
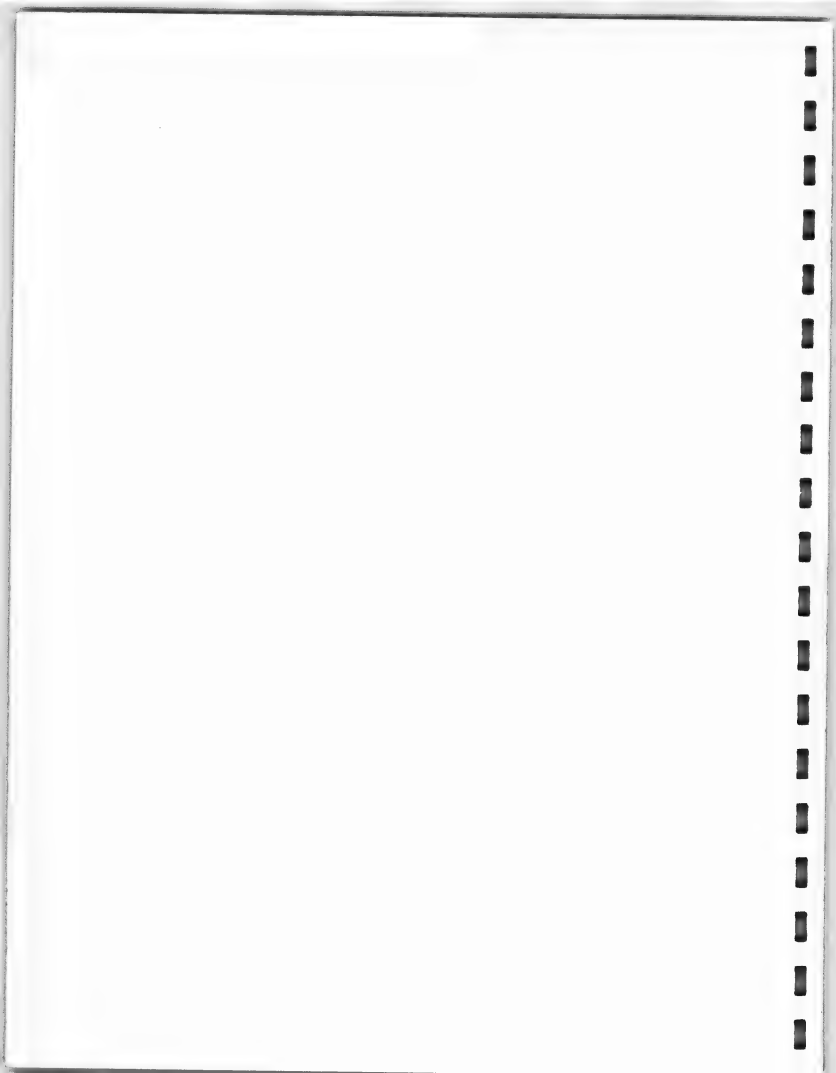


Table F-1. Floating Point Numbers

Number (in decimal)	Mantissa (HEX)	Exponent (HEX)	Register N+1 Register N Machine
21.0	540000	85	5400 0085
-.5	800000	7F	007F 8000
45×10^{-1}	480000	83	4800 0083
0.0	000000	00	0000 0000
1	400000	81	4000 0081
-1	800000	80	8000 0080
-89	A70000	87	A700 0087
(1) $.14693680 \times 10^{-38}$	400000	00	4000 0000
(2) $.17014116 \times 10^{+39}$	7FFFFF	FF	7FFF FFFF
(3) $-.14693863 \times 10^{-38}$	BFFFFF	00	BFFF FF00
(4) $-.170141182 \times 10^{+39}$	800000	FF	8000 00FF
3.14159265	6487ED	82	6487 ED82

- (1) Smallest Positive Value
 (2) Largest Positive Value
 (3) Smallest Negative Value
 (4) Largest Negative Value



APPENDIX G

ATAC-16M Master/Reset Start-Up Sequence

1. Interrupt System Initial Condition

- Interrupts are disabled (DSI state)
- The interrupt mask (IMR) is fully enabled (set to 1's)
- The interrupt priority register (IPR) is set to the lowest priority

2. Starting Address

ATAC-16M performs an indirect branch through location 1 (the start-up routine entry point address). If the control panel is connected to the ATAC system, a halt occurs with the starting address and first instruction of the start-up routine displayed. The register notation is as follows:

- (1) → Program counter
- ((1)) → Instruction execution register

If the control panel is not attached, the ATAC immediately begins execution of the program whose starting address is found in memory location 1.

IF YOU HAVE COMMENTS
OR SUGGESTIONS
ABOUT THIS PUBLICATION,
PLEASE LET US KNOW!

FOLD

FOLD

Publication Title _____

Date of Publication _____

Comments, suggestions _____

FOLD

FOLD

Your name _____

Title _____

Company _____

Address _____

FOLD AND STAPLE. NO POSTAGE
NECESSARY IF MAILED IN U.S.A

BUSINESS REPLY CARD

No postage necessary if mailed in the United States

Postage will be paid by

Applied Technology

A Division of Itel Corporation.
645 Almanor Avenue
Sunnyvale, California 94086

Attn.:
Dept. 1290

FIRST CLASS
PERMIT
No. 726
Sunnyvale,
Calif.



